Titre: Title:	Consistance et protection des données dans les systèmes collaboratifs par les méthodes formelles
Auteur: Author:	Moustapha Bande
Date:	2018
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Bande, M. (2018). Consistance et protection des données dans les systèmes collaboratifs par les méthodes formelles [Ph.D. thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/3695/

UNIVERSITÉ DE MONTRÉAL

CONSISTANCE ET PROTECTION DES DONNÉES DANS LES SYSTÈMES COLLABORATIFS PAR LES MÉTHODES FORMELLES

MOUSTAPHA BANDE
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIÆ DOCTOR
(GÉNIE INFORMATIQUE)
NOVEMBRE 2018

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée :

CONSISTANCE ET PROTECTION DES DONNÉES DANS LES SYSTÈMES COLLABORATIFS PAR LES MÉTHODES FORMELLES

présentée par : BANDE Moustapha
en vue de l'obtention du diplôme de : Philosophiæ Doctor
a été dûment acceptée par le jury d'examen constitué de :

Mme NICOLESCU Gabriela, Doctorat, présidente
Mme BOUCHENEB Hanifa, Doctorat, membre et directrice de recherche
M. KHOMH Foutse, Ph. D., membre
M. BENTAHAR Jamal, Ph. D., membre externe

DÉDICACE

À ma famille et mes parents.

REMERCIEMENTS

En premier lieu, je tiens à remercier fortement ma directrice de thèse Hanifa Boucheneb qui n'a ménagé aucun effort pour l'encadrement de mon travail tout au long de ces années d'études doctorales. Je ne saurai passer sous silence son immense apport tant au niveau des connaissances, du support humain et financier.

Je remercie également Hakima Ould-Slimane pour son implication dans mon encadrement et ses suggestions et commentaires d'amélioration de mes travaux de recherche.

Mes remerciements vont à Gabriela Nicolescu pour m'avoir fait l'honneur de présider ce jury de thèse et Jamal Bentahar pour avoir accepté de participer à mon jury à titre de membre externe, sans oublier Foutse Khomh pour son rôle de membre.

Toute ma profonde gratitude va à ma famille proche et lointaine, particulièrement à mon fils tant aimé Jamel Jabbar, qui a toujours été ma source de motivation pour aller de l'avant et terminer cette thèse.

Je tiens aussi à remercier mes collègues et aînés du laboratoire Veriform Aurel Randolph et Mohammad-Reza Gholami pour leurs conseils et bonne humeur.

Je remercie également mon ex-collègue Frédéric Correia pour ses conseils et soutien.

Je remercie très sincèrement toutes les personnes non distinctement nommées qui m'ont apporté leur soutien de toute sorte durant ce long voyage scientifique et humain.

RÉSUMÉ

Le développement de logiciels complexes ou de contenus multimédias implique de nos jours plusieurs personnes de plus en plus géographiquement dispersées à travers le monde qui collaborent à travers des systèmes d'édition collaborative. Le but principal de cette collaboration est l'amélioration de la productivité et la réduction du temps et des coûts de développement. L'un des défis majeurs de ces outils d'édition collaborative est d'assuer la convergence et la fiabilité des données partagées. Pour répondre à ce défi, plusieurs approches existent dans la littérature parmi lesquelles, nous avons l'approche multiversions (MV), l'approche des types de données commutatives répliquées (CRDT) et l'approche de la transformée opérationnelle (OT). La première se base sur le principe du *copier, modifier et fusionner* et utilise un serveur central chargé de la fusion des différentes copies provenant des sites participant à la collaboration. Les modifications effectuées par chaque site sur sa copie ne sont fusionnées à la copie centrale qu'à la demande de l'utilisateur. La difficulté majeure de cette approche est le coût lié au stockage des diverses versions sur le serveur, l'utilisation d'estampilles pour ordonner les opérations lors de la fusion. Ce qui la rend difficilement utilisable dans un contexte d'environnement distribué. La deuxième approche considère que toutes les opérations sont commutatives donc pouvant être exécutées dans un ordre quelconque. Quant à la dernière approche, elle s'appuie sur une transformation des opérations reçues des sites distants par rapport à celles qui leur sont concurrentes. Dans cette approche, un algorithme de transformation inclusive (IT) est utilisé afin d'assurer la convergence des copies, mais la plupart des algorithmes proposés dans la littérature ne satisfont pas les critères de convergence.

En plus de la cohérence, la fiabilité des données reste un autre défi dans les systèmes collaboratifs. Pour faire face à ce défi, plusieurs applications encapsulent, dans leur code source, des fonctionnalités transverses telles que celles de sécurité afin de répondre aux exigences de confidentialité et d'intégrité des données. Dans la littérature, la programmation orientée aspect (AOP) est l'une des approches utilisées afin d'assurer la modularité, la maintenabilité et la réutilisation des composants d'un logiciel. Une des difficultés de ce paradigme de programmation est l'assurance qu'une propriété de sécurité reste satisfaite après le tissage entre le programme de base et tous les aspects encapsulant les préoccupations transverses. Ce qui implique de trouver des techniques automatiques de vérification des propriétés de sécurité une fois le tissage fait. Dans le registre de la fiabilité des données, le contrôle d'accès joue un rôle prépondérant. Ainsi, en ce qui concerne le partage de contenus multimédias publiés sur le Web, il est nécessaire de collaborer pour les alimenter, mais un des défis majeurs est de les rendre fiables.

Dans la présente thèse, nous nous penchons sur les défis que sont la cohérence et la protection des données dans les systèmes collaboratifs dont un des domaines d'application est le

codage collaboratif. Pour la fiabilité des données dans le contexte de partage de contenus multimédias, nous considérons que le modèle de contrôle d'accès basé sur les attributs (ABAC) serait adapté à notre approche vu le caractère dynamique de la collaboration. Nous faisons l'hypothèse que les méthodes formelles peuvent nous aider à automatiquement et dynamiquement apporter la preuve de la fiabilité de nos données partagées. La cohérence est traitée dans un cas pratique de codage collaboratif où l'objet manipulé est une ligne de code d'un programme et les opérations considérées sont l'insertion et la suppression d'une ligne de code. L'approche utilisée est celle de la transformée opérationnelle où chaque utilisateur a sa copie de code source et les modifications effectuées par un site sont immédiatement propagées aux autres sites pour intégration. La présente thèse se présente sous forme de trois articles scientifiques qui traitent chacun d'une problématique particulière.

Dans le premier article, nous abordons la problématique de la fiabilité des données. Dans cet article, nous proposons l'utilisation du modèle de contrôle d'accès ABAC pour renforcer la sécurité d'une plateforme de création et de partage de contenus (comparable à Wikipédia). Nous proposons ensuite une preuve de la fiabilité du modèle de contrôle d'accès en utilisant le solveur "Satisfiability Modulo Theories" (SMT) $Z3$. Chaque règle de la politique de contrôle d'accès est transformée en contrainte puis passée au solveur pour vérifier sa satisfiabilité. Après vérification, toutes les règles de notre politique d'accès sont satisfaites.

Le deuxième article propose un mécanisme formel de vérification de propriétés notamment celles de sécurité dans un programme orienté aspects. En effet, ce paradigme attire la curiosité de la communauté des chercheurs mais peine à être adopté par la communauté des développeurs de logiciels. Nous nous sommes basés sur un exemple concret de programme de gestion de guichet automatique de banque. Notre mécanisme combine l'utilisation du graphe de flot de contrôle (CFG) du programme lors du tissage afin de déduire une formalisation à l'aide de solveurs SMT ($Z3$) pour valider la satisfaction des propriétés généralement encapsulées dans les aspects.

Le troisième article aborde la problématique de la convergence dans le codage collaboratif pour un environnement distribué. Nos investigations nous montrent que jusqu'ici les recherches se sont focalisées sur la proposition d'approches permettant la collaboration dans le développement de logiciels, mais peu d'entre elles ont proposé des algorithmes de transformation opérationnelle et la preuve qu'ils assurent la convergence dans ce domaine. Ainsi, nous proposons une fonction de transformation inclusive qui ajoute à la signature de l'opération d'insertion, un autre paramètre qui est la *position absolue* de la ligne de code au moment de son insertion, en faisant abstraction des lignes supprimées bien avant l'exécution de l'opération d'insertion. La preuve formelle de la convergence est démontrée dans cet article en utilisant l'outil *UPPAAL* et la preuve symbolique.

ABSTRACT

Complex software and Web content development involve nowadays multiple programmers located in different areas working together on the same development project using collaborative systems in order to achieve efficiency, improve productivity and reduce development time. One of the key challenge in such a development environment is ensuring the convergence and the reliability of the shared data or content. In the literature, many approaches have been proposed to face this challenge. Among those approaches, we have multi-version (MV), commutative replication data type (CRDT) and operational transformation (OT) approach. The first approach is based on the "copy, modify and merge" principle and uses a central server to merge the updates from the different sites participating in the collaboration. The local updates of a specific site are merged only on demand. The key drawback of this approach is the storage cost of the various versions on the server and the overhead due to the generation of stamps for the operations ordering. Thus, this drawback makes this approach difficult to use in the context of a distributed collaborative environment. The second approach preconizes that all the operations are commutative so that they can be executed in any given order. The latter approach is based on the transformation of all the operations received from the remote sites against their concurrent operations before being merged. In this approach, an inclusive transformation algorithm is used in order to ensure the convergence criteria. Unfortunately, most of the proposed algorithms in the literature do not satisfy the convergence criteria.

Beside the convergence, the reliability of the data remains another challenge in the collaborative systems. In order to face this challenge, many programs encapsulate crosscutting concerns (e.g. security, logging) for data confidentiality and integrity purposes. In the literature, aspect-oriented programming (AOP) is one the approaches used to better modularize the separation of concerns in order to make easier the maintenance and the reuse of the software components. However, one challenge of this paradigm is the insurance that a given property such as security one remains satisfied after the weaving of the base program and the aspects. Thus, we may find automated way to verify such security properties in the woven program. Concerning data reliability, access control is one of the major piece of the puzzle. Thus, in the Web content publication, one challenge is to collaborate in order to produce them and the other key challenge is to make them reliable.

This thesis is tackling consistency and reliability challenges in distributed collaborative systems, one of the scope being collaborative software development. For the reliability challenge, we assume that the attribute-based access control (ABAC) model is a serious candidate in the context of collaborative web content sharing due to the dynamic characteristic of the collaboration. We also assume that formal methods could help us to automatically and dynamically prove the reliability of the shared data. The consistency

challenge is handled by the practical case of a collaborative software development where the objects are lines of source code and operations are insert and delete a line of source code. We use the operational transformation approach to deal with the consistency challenge. Every user taking part in the collaboration has his copy of source code locally and the updates are automatically propagated to the other sites for merging. This thesis consists in three scientific articles, each of them deals with a specific challenge.

In the first article, we tackle the problem of data reliability. We propose the use of ABAC as an access control model to enforce the security of a framework of creation and sharing Web content such as Wikipedia. We then propose a proof of the reliability of our access control model using the satisfiability modulo theries (SMT) solver $Z3$. Each rule including in our access control policy is transformed as a constraint and added to the solver for satisfiability check. After the verification, we note that all the rules in the access control policy are satisfied.

In the second article, we propose a formal security property verification in an aspect-oriented program. Indeed, this paradigm arouses research community curiosity but struggles to be widely used by the software developers. We use a practical example based on the management of an automated teller machine (ATM) transactions. Our solution combines the generated control flow graph (CFG) of the woven program and a SMT $Z3$ model in order to confirm the satisfiability of some security properties encapsulated in the different aspects.

The third article addresses the consistency problem in the context of collaborative software development in a distributed environment. The investigations we have done so far in the literature show that people have focused on proposing tools and approaches that facilitate the collaboration but few studies have been done on finding an inclusive transformation algorithm and the proof that this algorithm ensures the convergence. Thus, we propose an IT function which adds a new parameter in the signature of the insert operation. This parameter is the *absolute position* of the line of code at the moment of its insertion without taking into account the number of deleted lines of code before the execution of the insert operation. We finally use the model checker *UPPAAL* and the symbolic proof to show that our IT function ensures the convergence.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

ABAC	Contrôle d'accès basé sur les attributs (Attribute-Based Access Control)
ACL	Liste de contrôle d'accès (Access Control List)
AOP	Programmation orientée aspects (Aspect-Oriented Programming)
ATCoPE	Any-Time Collaborative Programming Environment
ATM	Guichet automatique bancaire (Automatic Teller Machine)
CBAC	Contrôle d'accès basé sur le Contexte (Context-Based Access Control)
CFG	Graphe de flot de contrôle (Control Flow Graph)
C-List	Liste d'aptitudes (Capability List)
CoRED	Collaborative Real-time EDitor
CRDT	Type de données commutatives répliquées (Commutative Replicated Data Type)
CTL	Computational Tree Logic
CVS	Concurrent Version System
DAC	Contrôle d'Accès Discrétionnaire (Discretionary Access Control)
DAL	Dependency-based Automatic Locking
ICI	Idalho Collaborative Integrated development environment
IDE	Environnement intégré de développement (Integrated Development Environment)
IT	Transformation inclusive (Inclusive Transformation)
LES	Labelled Event Structures
LTL	Linear Temporal Logic
LTSA	Labeled Transition System Analyzer
MAC	Contrôle d'accès mandataire (Mandatory Access Control)
MV	MultiVersion
OT	Transformée opérationnelle (Operational Transformation)
P2P	Peer-to-Peer
PCDU	PointCut-based Def-Use
RBAC	Contrôle d'accès basé sur les rôles (Role-Based Access Control)
RECIPE	REaltime Collaborative Interactive Programming Environment
SAT	SATisfiabilité (SATisfiability)
SMT	Satisfiability Modulo Theories
SVN	SubVersioN
TBAC	Contrôle d'accès basé sur les tâches (Task-Based Access Control)
TMAC	Contrôle d'accès basé sur les équipes (TeaM-based Access Control)
TT-RBAC	Contrôle d'accès basé sur les équipes et les tâches (Team and Task Based Access Control)

YATA Yet Another Transformation Approach

CHAPITRE 1 INTRODUCTION

De nos jours, avec l'ascension fulgurante des technologies de l'information et de la communication, il est devenu indéniable pour les entreprises géographiquement dispersées un peu partout sur la planète d'adopter une autre façon de travailler et de partager l'information. Cela a amené les concepteurs de systèmes informatiques à s'intéresser davantage à la mise en place de systèmes collaboratifs. Les utilisateurs voient en ces environnements un moyen de partager plus efficacement leurs ressources et de réaliser des tâches communes afin de gagner en temps d'exécution et de réduire les coûts liés au temps de calcul. En effet, un système collaboratif ou outil de collaboration a pour but d'aider un groupe de personnes travaillant sur un même projet ou une même tâche à collaborer à travers un réseau informatique, afin d'atteindre un objectif préalablement ciblé. Comme exemple, nous pouvons citer une panoplie d'applications telles que le codage collaboratif, l'encyclopédie Wikipédia, la vidéo ou l'audio conférence, le partage de documents, l'apprentissage à distance, etc.

La plupart des systèmes collaboratifs existants ont été conçus dans le souci de rendre disponibles les ressources partagées, mais aussi d'être flexibles afin de faciliter la collaboration à tous ceux qui s'associent pour réaliser des tâches communes, sans leur imposer des contraintes majeures. En effet, dans la littérature, peu d'efforts ont été consacrés à la mise en place de mécanismes de sécurité dans ces systèmes afin d'introduire un certain contrôle sur les activités des membres collaborateurs. Autrement dit, il y a très peu de mécanismes de sécurité mis en œuvre afin d'assurer la fiabilité des activités collaboratives.

En effet, l'information, qui est considérée comme l'une des principales richesses des entreprises, devient très sensible et sa protection de ce fait constitue un enjeu primordial. Les systèmes d'information des entreprises impliquent des informations et des ressources caractérisées par différents degrés de sensibilité. Plus précisément, la moindre faille de sécurité dans le système d'information peut engendrer une divulgation d'informations, un déni de service ou une utilisation malveillante des ressources par des personnes n'ayant pas l'autorisation d'y accéder. Les conséquences de ces failles en sont souvent graves : le fait le plus marquant a été, par exemple, les révélations faites en 2010 par Wikileaks sur les rapports diplomatiques américains. Un autre exemple de conséquence lourde est l'accès aux dossiers de patients d'un hôpital par des personnes non autorisées entraînant la divulgation des informations à caractère personnel qui s'y trouvent.

Par conséquent, afin d'assurer l'intégrité, la confidentialité et la disponibilité des données, il est impératif de définir des politiques et d'instaurer des mécanismes de sécurité, afin de fixer les règles (les protocoles) selon lesquelles le partage des ressources est fait de manière sécurisée et fiable.

Dans la littérature, plusieurs approches de contrôle d'accès ont été proposées. Les modèles existants tels le modèle de contrôle d'accès discrétionnaire (DAC), le modèle de contrôle d'accès mandataire (MAC) ainsi que le modèle de contrôle d'accès basé sur les rôles (RBAC) sont pour la plupart, destinés à des systèmes statiques. Il existe cependant des modèles qui ont pris en compte le caractère contextuel des systèmes de collaboration. En effet, plusieurs dérivés du modèle RBAC ont été proposés pour pallier les difficultés rencontrées par le modèle RBAC traditionnel par rapport aux systèmes collaboratifs. Ces difficultés sont entre autres (i) l'incapacité de définir un contrôle fin sur des utilisateurs individuels dans certains rôles et sur certaines instances d'objets ; (ii) La rigidité du contrôle, car RBAC ne prend pas en compte l'impact contextuel (il est considéré comme un modèle passif de sécurité).

À ce jour, la majorité des modèles de contrôle d'accès proposés sont soit définis de manière informelle donc, sujets à des spécifications ambigües ou incomplètes, soit s'attaquent à une application spécifique. Ce qui explique qu'ils ne conviennent pas en tant que modèles de référence aux autres systèmes collaboratifs. Par conséquent, pour participer activement aux systèmes de collaboration modernes, les utilisateurs, les groupes d'utilisateurs et les membres d'organisations doivent être capables de partager des données spécifiques avec les autres membres de la collaboration, tout en assurant que les ressources partagées sont à l'abri d'accès inappropriés. De telles collaborations peuvent, de manière dynamique, changer de participants et de relations de confiance pendant le cycle de vie. La nature dynamique de ces environnements de collaboration et les entités hétérogènes impliquées engendrent des grands défis de sécurité qui demandent une nouvelle approche technique de définition des politiques de contrôle d'accès et leur application avec en sus un défi d'équilibrage des objectifs liés à la réduction de la charge de calcul et du contrôle d'accès pour l'information partagée.

Dans le présent chapitre, nous présentons dans un premier temps, quelques concepts de base permettant de mieux comprendre cette thèse. Ensuite, nous abordons successivement les éléments de la problématique, les objectifs de recherche et les principales contributions de notre thèse au monde de la recherche. Enfin, nous concluons ce chapitre par le plan de la thèse.

1.1 Définitions et concepts de base

Dans cette section, nous présentons quelques concepts liés à la programmation collaborative dans un environnement distribué incluant la notion de réplication et de convergence, la programmation orientée aspect (AOP) et les preuves formelles. Ces concepts sont la base de notre travail et sont utilisés tout au long de la thèse.

1.1.1 Programmation collaborative

Le développement de logiciels de grands systèmes de nos jours est un processus hautement collaboratif dans lequel le travail de groupe est essentiel. À travers ce processus, les programmeurs, localisés géographiquement à travers le monde, travaillent ensemble sur un même projet de développement de logiciel en utilisant exclusivement l'une des deux stratégies : éditer une copie unique du code en se basant sur un environnement intégré de développement (IDE) ou éditer des copies distinctes en parallèle en intégrant leurs efforts à travers un système de contrôle de version de code source. Dans la première stratégie, plusieurs programmeurs modifient les mêmes objets (fichiers, répertoires du même code source) et fusionnent manuellement leurs modifications en utilisant un système de contrôle de version tel que "Subversion" (SVN) (Pilato et al., 2008) ou "Concurrent version system" (CVS) (Berliner et al., 1990). Dans la dernière, chaque programmeur a sa copie de code source et les modifications apportées sont automatiquement propagées aux autres membres de la collaboration sans la nécessité d'avoir un serveur central. Ce genre d'édition collaborative peut être *synchrone* ou *asynchrone*. Elle est dite *synchrone* si les objets partagés sont édités au même moment, et *asynchrone* si l'édition des objets partagés se déroule à des moments différents par les collaborateurs. Pour ce qui concerne l'édition synchrone, chaque collaborateur possède sa copie locale de l'objet partagé appelée "réplique". Chaque membre de la collaboration peut manipuler sa copie en lecture comme en écriture et y générer des opérations. Les opérations générées localement sont ensuite propagées aux autres membres (sites distants) de la collaboration pour leur prise en charge à travers un mécanisme de propagation des modifications appelé "réplication d'opérations".

La réplication

Chaque utilisateur participant à la collaboration a accès au code (programme) partagé soit à travers un serveur central où il est hébergé ou un système distribué dans lequel chacun a sa propre copie localement. Dans le dernier cas, toute modification (opération d'insertion ou de suppression) faite localement par un des sites est propagée aux autres sites pour intégration. Chaque copie ou réplique doit refléter les opérations générées par tous les sites en collaboration afin d'avoir des copies identiques. Les techniques traditionnelles de réplication tentent de maintenir une seule copie cohérente et donnent l'illusion aux utilisateurs d'avoir une seule copie disponible de l'objet partagé (Herlihy and Wing, 1990). Pour atteindre cet objectif selon ces techniques, l'accès à la copie de l'objet est bloqué jusqu'à ce qu'elle soit à jour. Il s'agit dans ce cas de réplication *pessimiste*. Ces techniques fonctionnent bien dans un environnement de réseaux locaux où les temps de latence sont moindres. Par contre, dans des environnements hautement collaboratifs où les modifications sont nombreuses, leur disponibilité et leur productivité diminuent au fur et à mesure que le nombre de sites augmente. Un autre groupe de techniques de partage

efficient des données dans un environnement de réseau étendu est la réplication *optimiste*. Ce qui différencie la réplication optimiste de la pessimiste, c'est leur approche du contrôle de la concurrence. Tandis que l'approche pessimiste coordonne les répliques pendant les accès et les bloque durant une mise à jour, l'approche optimiste quant à elle permet aux utilisateurs d'accéder à l'objet partagé sans synchronisation en faisant l'hypothèse qu'il y aura rarement d'erreurs. Dans ce cas, les modifications sont propagées aux autres sites et les conflits occasionnels sont gérés après leur apparition. Les techniques de réplication optimistes améliorent la disponibilité de la donnée, sont flexibles et s'adaptent à des environnements d'un nombre très élevé de répliques, car elles requièrent peu de synchronisation entre les sites ; ce qui permet aux sites et utilisateurs de rester autonomes (Petersen et al., 1997).

Notion de convergence

La génération d'opérations sur les sites et leur intégration est une succession d'événements ayant un impact sur chaque réplique située dans les différents sites. Ce qui crée un changement d'état des répliques chaque fois qu'une opération est générée et prise en charge. La notion de convergence stipule qu'après que toutes les opérations générées ont été appliquées sur les répliques indépendamment de l'ordre de leur réception, les copies sont identiques et l'intention des utilisateurs demeure respectée.

1.1.2 Programmation orientée aspect

Afin d'assurer les qualités de modularité, de maintenabilité et de réutilisabilité des composants d'un système logiciel, des approches de découpage orienté aspect ont été proposées dans la littérature. Kiczales et al, dans (Kiczales et al., 1997), définissent le paradigme de programmation orientée aspect comme étant la séparation des préoccupations (fonctionnalités) entre le programme de base et celui regroupant les aspects. Le premier est réservé aux fonctionnalités principales tandis que le second regroupe les préoccupations transverses. Une préoccupation est dite transverse si elle impacte plusieurs modules dans son implémentation. Comme exemple de préoccupation transverse, on peut citer certaines fonctionnalités interclasses (journalisation) ou des modules de sécurité ou de persistance de données. Ce paradigme a apporté d'autres concepts dont les principaux sont :

— **Point de jonction (Join point)**. C'est un point défini à l'intérieur d'une classe et qui sert de point d'attache à une préoccupation au cours de l'exécution du programme (ex. appel d'une méthode, exécution d'une méthode, levée d'une exception, invocation d'un constructeur, etc.) ;

— **Point de coupure (Point cut)**. C'est un regroupement de points de jonction qui sert à définir la structure transversale d'un aspect. C'est une portion de code défini dans un aspect ;

— **Advice**. C'est un bloc de code représentant une action prise par l'aspect à un point de jonction en particulier. Il est implémenté comme une méthode de la classe aspect. Mais il est exécuté *avant*, *après* ou *à la place* des points de jonction liés à un point de coupure de l'aspect ;

— **Tissage**. C'est le processus permettant d'exécuter l'advice approprié au niveau de chaque point de jonction. Ceci est réalisé par un composant appelé *tisseur d'aspect (aspect weaver)*. Ce composant est chargé de greffer les aspects sur le code de base (cible). Le programme qui résulte de cette opération est appelé *programme tissé (woven program)*. Ce processus est schématisé dans la figure 1.1 dans (Baltus, 2001).

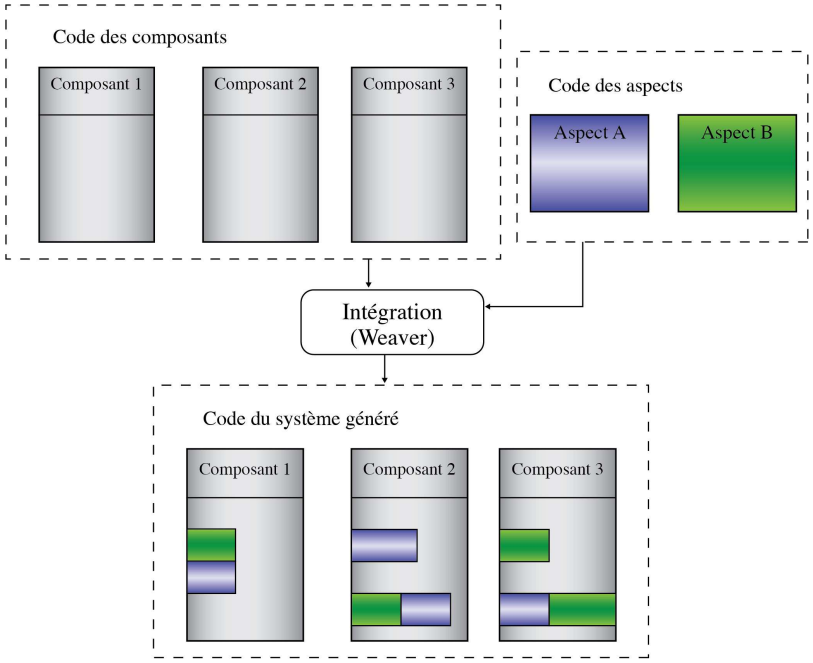


Figure 1.1 Processus de tissage

1.1.3 Solveurs "Satisfiability Modulo Theories" (SMT)

Les solveurs SMT sont des outils permettant de vérifier la satisfiabilité de formules de premier ordre contenant des opérations de diverses théories telles que les booléens, les vecteurs de bit, l'arithmétique, les tableaux, les types de données récursives. Les solveurs SMT sont une extension des solveurs de problèmes de satisfiabilité booléenne (SAT) auxquels on a rajouté une procédure de décision pour chaque théorie. Ils possèdent beaucoup de domaines d'applications tels que la vérification logicielle et matérielle, la résolution de contraintes, l'ordonnancement, la planification, la génération de cas de tests ou encore la sécurité informatique. La satisfiabilité n'est autre que le fait de pouvoir déterminer si une formule donnée exprimant une contrainte a un modèle ou une solution. Parmi les solveurs SMT, nous nous sommes intéressés à $Z3$: un outil proposé par Microsoft (De Moura and

Bjørner, 2008) qui permet de vérifier la satisfiabilité de formules logiques sous une ou plusieurs théories et qui se base sur la logique de premier ordre. Une formule ϕ est dite *satisfaite* s'il existe un modèle ou une interprétation qui rend ϕ vraie. La formule est dite *valide*, si elle est vraie pour toutes les théories. Par exemple, la formule $x + y < 3$ est *satisfaite* sous l'interprétation $x = 1$ et $y = 1$.

1.2 Éléments de la problématique

Dans cette section, nous faisons une critique des modèles de contrôle d'accès existants dans la littérature relativement à notre domaine de recherche afin de mieux situer la problématique du contrôle d'accès dans les systèmes collaboratifs. Dans (Tolone et al., 2005), les auteurs synthétisent les exigences de contrôle d'accès que ces environnements devraient satisfaire :

- le contrôle d'accès doit être appliqué et imposé au niveau d'une plateforme distribuée ;
- les modèles de contrôle d'accès doivent être génériques et permettre aux droits d'accès d'être configurés pour répondre aux besoins d'une large variété de tâches coopératives et de modèles d'entreprises. Ce qui implique que de tels modèles doivent être assez expressifs pour spécifier de manière efficace les droits d'accès basés sur des informations variées (exemple : les rôles, le contexte...etc.) ;
- le contrôle d'accès pour la collaboration exige une plus grande évolutivité en ce qui concerne la quantité d'opérations comparativement aux modèles traditionnels mono-utilisateurs. En effet, l'ensemble d'opérations partagées est plus diversifié dans les environnements collaboratifs comparés à celui des systèmes traditionnels (mono-utilisateurs) ;
- les modèles de contrôle d'accès doivent être capables de protéger l'information et les ressources, indépendamment de leur degré de granularité, sur des niveaux variés. En effet, ils doivent offrir la capacité aussi bien de fournir une protection forte pour les environnements partagés et les objets de types variés que de permettre un contrôle d'accès fin sur les objets individuels ainsi que leurs attributs ;
- les modèles de contrôle d'accès doivent faciliter un accès transparent aux utilisateurs autorisés et une exclusion stricte des utilisateurs non autorisés de manière flexible de façon à ne pas contraindre la collaboration ;
- les modèles de contrôle d'accès doivent permettre une spécification de haut niveau des droits d'accès pour mieux gérer la complexité de plus en plus croissante induite par la collaboration. Les modèles de contrôle d'accès pour la collaboration doivent être dynamiques, c'est-à-dire qu'il doit être possible de modifier les politiques durant l'exécution en fonction de l'environnement ou de spécifier de nouvelles contraintes au cours de la collaboration ;
- les coûts de performance et des ressources doivent être maintenus dans des limites acceptables.

En analysant les exigences que doivent remplir les modèles de contrôle d'accès dans les environnements collaboratifs, nous pouvons constater que la plupart des modèles ne remplissent pas toutes les exigences citées ci-dessus. En fait, ils ont été spécialement conçus

pour répondre à des besoins particuliers. Par conséquent, ces modèles ne pourraient répondre aux besoins de tout système de collaboration. En ce qui concerne le codage collaboratif, la plupart des environnements de travail ont été conçus dans le but de permettre l'édition collaborative alors que d'autres, en plus de cette fonctionnalité, se sont focalisés sur l'interaction entre les différents acteurs de la collaboration par la sensibilisation, les "chats", la messagerie et aussi la prise de décision par consensus concernant la synchronisation des modifications. Dans la littérature, à notre connaissance peu (voire pas –du moins d'une manière explicite-) de travaux se sont réellement penchés sur l'applicabilité de la sécurité dans les systèmes de programmation collaborative. La contribution de (Briffaut et al., 2009) est la plus pertinente que nous avons recensée dans ce contexte. Les auteurs y proposent la génération de politiques de sécurité basées sur le modèle RBAC pour les applications collaboratives écrites en Java. Ce travail se limite à offrir à l'utilisateur le choix du type de politique qu'il souhaiterait appliquer au moment de l'installation de l'application. Nous rappelons que la sécurité de l'information en général et du code source en particulier, constitue un enjeu très important pour les entreprises de développement de logiciels. Ainsi, les projets de développement de logiciels qu'ils soient ouverts (open source) ou non doivent intégrer des mécanismes efficaces et fiables qui permettent de garantir un accès sécurisé à toute partie du code source par des programmeurs autorisés. Le principal défi dans ces systèmes est non seulement de pouvoir concilier un environnement collaboratif flexible et facile d'utilisation tout en y appliquant des mécanismes de sécurité protégeant l'accès aux ressources partagées, mais aussi de trouver des moyens permettant de détecter automatiquement les incohérences et anomalies liées aux erreurs de programmation ou d'architecture du système logiciel.

Dans les systèmes d'édition collaborative, il survient la plupart du temps des divergences dans les répliques des différents sites dues principalement aux nombreuses mises à jour des copies locales et leur propagation aux autres sites. Cette problématique de divergence a fait l'objet de plusieurs recherches au cours des deux dernières décennies pour trouver des algorithmes de transformation ou d'intégration pouvant garantir la convergence des répliques lors d'une collaboration dans un environnement distribué. Ainsi, l'approche de la transformée opérationnelle développée par (Ellis and Gibbs, 1989; Sun and Ellis, 1998; Sun et al., 1998; Vidot et al., 2000) reste la principale approche qui tente de donner une solution à la problématique de la divergence. L'idée principale de l'OT est la transformation des opérations reçues des autres sites avant de les exécuter localement (Li and Li, 2004). Pour se faire, les deux composants nécessaires selon (Boucheneb and Imine, 2009) sont : l'algorithme d'intégration et la fonction IT. Le premier s'occupe de la propagation des opérations aux autres sites et de leur exécution lors de leur réception. C'est aussi à ce composant que revient la charge de détecter les opérations concurrentes et de leur ordre d'application après transformation. Le dernier se charge de transformer une opération reçue relativement à d'autres opérations qui lui sont concurrentes et exécutées

précédemment. Le principal défi est non seulement la mise en place d'un mécanisme approprié de détection de la concurrence, mais aussi une fonction de transformation inclusive garantissant la cohérence c'est-à-dire le respect de l'intention (principe de causalité) et la convergence des répliques. Pour respecter la convergence dans l'approche OT, deux propriétés doivent être satisfaites selon (Ressel et al., 1996). Il s'agit de :

— la propriété $TP1$: elle définit une *identité d'état* et assure que lorsqu'on a deux opérations o_1 et o_2 concurrentes, le résultat de l'exécution de o_1 avant o_2 est le même que si on exécutait o_2 avant o_1. En d'autres termes, la séquence d'opérations obtenue en considérant o_1 suivie de la transformation de o_2 par rapport à o_1 est équivalente à la séquence obtenue en considérant o_2 suivie de la transformation de o_1 par rapport à o_2 ;

$$[o_1; IT(o_2, o_1)] \equiv [o_2; IT(o_1, o_2)] \tag{1.1}$$

avec $IT(o_2, o_1)$ la fonction de transformation inclusive de o_2 par rapport o_1.

— la propriété $TP2$: définit une *identité d'opérations* et assure que la transformation d'une opération o_3 par rapport à une séquence équivalente d'opérations différentes ne dépend pas de l'ordre dans laquelle les opérations ont été transformées.

$$IT^*(o_3, [o_1; IT(o_2, o_1)] \equiv IT^*(o_3, [o_2; IT(o_1, o_2)] \tag{1.2}$$

avec IT^* l'extension de IT comme étant la composition de la fonction de transformation inclusive.

1.3 Objectifs de recherche

Cette thèse a pour cadre général la consistance et la protection des données dans les systèmes collaboratifs en utilisant les méthodes formelles. Plus précisément, il s'agit :

1. d'analyser les modèles de contrôle d'accès existants afin d'adapter leurs mécanismes aux exigences spécifiques des systèmes collaboratifs ;
2. de proposer des applications effectives de ces mécanismes de sécurité dans une plateforme de partage de contenus informatifs ;
3. de concevoir un algorithme de transformation inclusive garantissant la convergence et l'appliquer au codage collaboratif ;
4. et de proposer une approche de vérification formelle des propriétés de sécurité dans le contexte de la programmation orientée aspect.

1.4 Esquisse méthodologique

Pour atteindre nos objectifs spécifiques ci-dessus cités, un travail préliminaire consiste à faire une revue approfondie de la littérature sur les modèles de contrôle d'accès existants et mesurer leur impact sur les environnements collaboratifs. À l'issue de cette analyse et pour satisfaire le premier objectif spécifique, nous définirons, par rapport aux exigences déjà énoncées par (Tolone et al., 2005), des exigences compétitives, conflictuelles, dynamiques, évolutives pour le contrôle d'accès des systèmes collaboratifs. À l'issue de cette étape, il s'agit d'établir des procédés permettant d'imposer ces exigences aux applications collaboratives ciblées. La recherche de solutions pour l'application de ces exigences revient à la prise en compte du caractère dynamique des systèmes collaboratifs. Ces derniers doivent être faciles d'utilisation, flexibles pour les usagers tout en étant « suffisamment » sécurisés afin de répondre à l'objectif 2. Par conséquent, lors de la conception de nos mécanismes, il faudra faire en sorte que ces deux buts -parfois contradictoires- soient conciliés et équilibrés. En effet, il s'agit d'éviter d'imposer les contrôles d'accès au détriment de la flexibilité de ces systèmes (dont la facilité d'utilisation est à l'origine de cet engouement populaire). Quant à l'objectif 3, la démarche à suivre pour l'atteindre consiste à analyser les algorithmes de transformation inclusive applicables au contexte du codage collaboratif ; et proposer une autre IT facile d'utilisation et qui assure la convergence des répliques. Finalement, pour ce qui est de l'objectif 4, nous ferons l'hypothèse que l'utilisation de la programmation orientée aspect peut faciliter la définition et l'application des politiques de contrôle d'accès dans ces environnements de collaboration. Et pour atteindre ce dernier objectif, nous utiliserons les méthodes formelles pour une vérification automatique de propriétés transversales encapsulées dans les aspects.

1.5 Plan de la thèse

Le reste de la présente thèse se structure comme suit. Le chapitre 2 présente une revue de littérature sur des modèles de contrôle d'accès existants et mesurer leur impact sur les environnements collaboratifs, le codage collaboratif incluant la convergence des répliques.

Le chapitre 3 dont le titre est *Formal Proof of WiseShare : A Collaborative Environment for Knowledge Sharing Governed by ABAC Policies* traite de l'application des méthodes formelles pour prouver la fiabilité d'un modèle de contrôle d'accès dans un contexte collaboratif de développement et le partage de contenus multimédia. La solution dans cet article utilise le modèle de contrôle d'accès ABAC pour renforcer la sécurité de ce genre d'environnement afin d'avoir des contenus plus fiables tout en gardant les contributions ouvertes à tous. En plus, la preuve de la fiabilité du modèle est faite en utilisation le solveur SMT *Z*3. Cet article a été soumis pour publication au journal *Information Security Journal : A Global Perspective*.

Pour ce qui est du chapitre 4 dont le titre est *A Formal Property Verification for Aspect-Oriented Programs in Software Development*, il est consacré à la vérification formelle de propriétés et fonctionnalités d'un code source AOP afin de s'assurer de leur cohérence. Notre approche est basée sur la génération d'un graphe de flot de contrôle à partir du code source tissé issu du programme de base et des aspects. C'est à partir de ce graphe que nous utilisons le solveur SMT $Z3$ pour la vérification de propriétés et fonctionnalités une fois le tissage terminé. Cet article est accepté pour publication à la conférence *20th International Conference on Aspect-Oriented Software Systems, Design, Development and Programming*.

Le chapitre 5 intitulé *Consistent Operational Transformation applied to Collaborative Coding Editors* propose un algorithme de transformation inclusive assurant les propriétés de convergence et appliqué à un système de codage collaboratif dans un environnement réparti. Nous avons démontré en utilisant les méthodes formelles que l'IT proposée préserve la convergence à travers les propriétés $TP1$ et $TP2$. Cet article est soumis pour publication au journal *Journal of Systems and Software*.

Ensuite, le chapitre 6 présente une discussion générale sur les aspects méthodologiques et les résultats obtenus lors de nos travaux de recherche. Enfin, le chapitre 7 conclut la présente thèse en faisant ressortir une synthèse des travaux, une critique de notre travail et les travaux de recherche futurs.

CHAPITRE 2 REVUE DE LITTÉRATURE

Cette partie examine les différentes approches relatives à notre sujet de recherche. Nous donnerons un aperçu général des modèles de contrôle d'accès traditionnels et ceux proposés dans les environnements collaboratifs. Par la suite, nous introduirons d'autres applications des systèmes collaboratifs dans le domaine du développement de logiciel.

2.1 Modèles traditionnels de contrôle d'accès

Le contrôle d'accès a toujours été une problématique majeure dans le monde informatique d'une manière générale, et principalement dans les systèmes d'exploitation et de gestion de bases de données. Il a pour objectif de protéger les ressources contre les accès inappropriés tout en autorisant les accès des utilisateurs légitimes. Lors des premières études de la recherche sur les contrôles d'accès, deux approches avaient été privilégiées : le contrôle d'accès discrétionnaire et le contrôle d'accès mandataire. Mais une troisième qui est le modèle de contrôle d'accès basé sur les rôles a vu le jour et en est le plus populaire de tous de nos jours.

2.1.1 Le contrôle d'accès discrétionnaire (DAC)

Le DAC est basé sur l'identité du demandeur et sur les règles d'accès stipulant ce que les demandeurs sont autorisés ou non à faire. Ce type de contrôle d'accès est appelé « discrétionnaire », car il permet aux utilisateurs de passer leurs privilèges aux autres utilisateurs, étant entendu que l'allocation et la révocation des privilèges sont contrôlées par une politique administrative. Le modèle DAC est beaucoup plus flexible et est utilisé dans la plupart des cas dans le secteur gouvernemental et commercial. Cependant, le modèle présente quelques faiblesses : (Hu et al., 2006)

— l'information peut être copiée d'un objet à un autre. Ainsi il n'y a aucune réelle assurance sur le flux d'information dans le système ;

— il n'y a pas de restriction appliquée à l'utilisation de l'information quand un utilisateur la reçoit ;

— les privilèges d'accès aux objets sont décidés par le propriétaire de l'objet, en lieu et place d'une politique du point de vue système qui reflète les exigences de la sécurité de l'organisation.

2.1.2 Le contrôle d'accès mandataire (MAC)

Les politiques de contrôle d'accès mandataires appliquent le contrôle d'accès sur la base de régulations mandatées par une autorité centrale. Ce modèle a été premièrement for-

malisé par (Bell and LaPadula, 1973). Ensuite, Sandhu, dans (Sandhu, 1993), a introduit un modèle qui renferme l'essentiel du modèle de Bell et LaPadula. Dans les deux cas, l'accès est accordé selon l'étiquette de sécurité du sujet et de l'objet accédé. La forme la plus connue des politiques mandataires est la politique de sécurité multi niveaux basée sur les classifications des sujets et des objets dans le système. Les objets sont des entités passives stockant l'information tandis que les sujets sont des entités actives qui demandent à accéder aux objets. Il existe une distinction entre les sujets d'une politique mandataire et les sujets d'autorisation des politiques discrétionnaires. En effet, tandis que les sujets d'autorisation correspondent aux utilisateurs ou groupes d'utilisateurs, les politiques mandataires font une distinction entre utilisateurs et sujets. Les utilisateurs sont des êtres humains qui peuvent accéder au système, pendant que les sujets sont des processus opérant au nom des utilisateurs. Cette distinction permet à la politique de contrôler les accès indirects (tels que les fuites ou les modifications) causés par les processus. La sémantique associée aux classes d'accès assignées aux objets et sujets à travers l'utilisation d'une politique mandataire dépend du fait que la classification est prévue pour une politique de confidentialité ou d'intégrité.

2.1.3 Le modèle Bell-LaPadula

C'est un modèle de contrôle d'accès mandataire basé sur la confidentialité. Il permet de contrôler le flux direct et indirect de l'information dans le but de prévenir les fuites des sujets non autorisés. Dans ce modèle, le niveau de sécurité de la classe d'accès associé à un objet reflète la sensibilité de l'information contenue dans l'objet. Les requêtes d'accès faites par un sujet à un objet sont contrôlées en tenant compte de la classe de l'objet et du sujet et sont accordées seulement si un certain nombre de relations relatives à l'accès demandé sont satisfaites. Le modèle peut être représenté par une machine à états où chaque état est un triplet $(b, M_s o, f) \in B \times M \times F$ où : $B = S \times O \times A$ est l'ensemble des opérations d'accès en cours où A = exécuter, lire, ajouter, écrire ; M est l'ensemble des matrices de permissions d'accès $M = (M_s o)(s \in S, o \in O)$. $M_s o$ associe les objets aux sujets en indiquant les droits qui s'y appliquent. $F \subset L_s \times L_c \times L_o$ est l'ensemble des niveaux de sécurité. Dans un état donné, fS est le plus haut niveau de sécurité qu'un sujet peut avoir ; fC est le niveau courant de chaque sujet ; fO est la classification de l'objet. Pour protéger la confidentialité de l'information, deux principes ont été formulés par (Bell and LaPadula, 1973). La figure 2.1 montre un aperçu de ces deux principes qui sont énoncés comme suit :

— aucune lecture ascendante (no-read-up en anglais) : il est permis à un sujet un accès en lecture à un objet si et seulement si la classe d'accès du sujet domine celle de l'objet ; $\forall (s, o, a) \in b, a \in \{lire, ecrire\}\, (fO(o) \leq fS(s))$.

— aucune écriture descendante (no-write down) : un sujet peut avoir un accès en écriture à un objet si et seulement si la classe d'accès de l'objet domine celle du

sujet. $\forall(s, o, a) \in b, a \in \{ajouter, ecrire\}\ (fc(o) \leq fo(s))$.

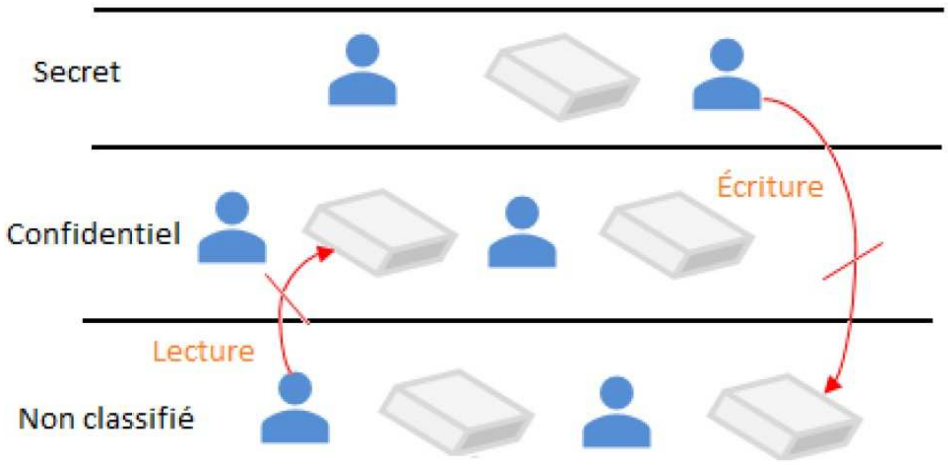


Figure 2.1 Principles "aucune lecture ascendante et aucune écriture descendante"

L'inconvénient majeur du modèle de Bell et LaPadula est le problème lié à la modification du niveau de sécurité des sujets et objets. Une révision du modèle a été faite par (Bell, 1974) et stipule qu'il y a une distinction entre le niveau de sécurité assigné à un sujet et son niveau courant. Ce qui introduit plus de flexibilité dans le contrôle. Un autre inconvénient du modèle est qu'il ne traite pas de l'intégrité des données.

2.1.4 Le modèle Biba

C'est un modèle de contrôle mandataire basé sur l'intégrité de l'information. Reprenant les principes fondamentaux de Bell et LaPadula, (Biba, 1977) a proposé une double politique pour sauvegarder l'intégrité, ce qui permet de contrôler le flux d'information et d'empêcher des sujets de modifier indirectement l'information qu'ils ne peuvent pas écrire. Les sujets et objets sont classés de manière hiérarchique. La classification est composée de trois éléments qui sont i) crucial (C), ii) important (I) et iii) indéterminé (U) ; et la relation entre les éléments s'exprime par $C > I > U$. Le niveau d'intégrité associé à un objet reflète le degré de confiance d'un utilisateur quant à l'insertion, la modification ou la suppression de l'information. La figure 2.2 montre comment est géré le flux d'information pour assurer l'intégrité des données dans ce modèle. Les principes fondamentaux du modèle Biba sont contraires à ceux de Bell et LaPadula et s'expriment comme suit :

— aucune écriture ascendante (no-write-up en anglais) : un sujet ayant un niveau donné d'intégrité ne doit pas pouvoir écrire dans un objet qui a un niveau d'intégrité plus élevé. Ceci est connu sous le nom de l'axiome de l'intégrité simple.

— aucune lecture descendante (no-read-down) : un sujet ayant un niveau d'intégrité donné ne doit pas pouvoir lire un objet qui a un niveau d'intégrité inférieur. Ceci est connu sous le nom de l'axiome de l'intégrité "*" (étoile).

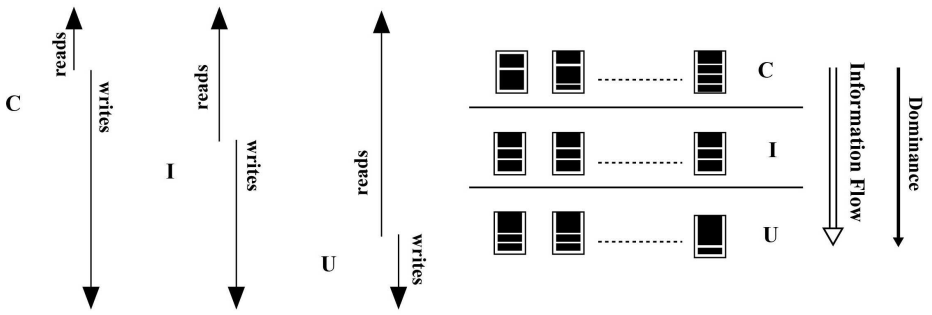


Figure 2.2 Flux d'information assurant l'intégrité des données

Les inconvénients du modèle Biba sont :

— il ne supporte pas la confidentialité ;

— il ne supporte pas non plus l'octroi et la révocation d'autorisations ;

— des problèmes surviennent quand le modèle Biba est utilisé dans un environnement réseau.

Les limites majeures des politiques de contrôle d'accès mandataire tiennent au fait qu'elles supportent un ensemble fixe de classes de sécurité, un ensemble fixe d'opérations sur les objets et qu'il est permis uniquement aux administrateurs de modifier la politique de contrôle d'accès.

2.1.5 Le modèle de la muraille de Chine

Ce modèle est le fruit des travaux de (Brewer and Nash, 1989) dans le but de s'attaquer aux questions de conflits d'intérêts relatifs aux activités de consultance entre les banques et les autres disciplines financières (par exemple, un consultant individuel ne doit pas avoir en sa possession des informations relatives à deux banques ou deux entreprises pétrolières). Cependant, contrairement au modèle de Bell-LaPadula, l'accès aux données n'est pas contraint par la classification des données, mais plutôt par les données auxquelles les sujets ont déjà accès. Le modèle est basé sur une organisation hiérarchique des objets détaillée ci-dessous :

— les objets du niveau le plus bas sont les données de toutes les compagnies ;

— les objets du niveau intermédiaire regroupent l'ensemble des données qui concernent la même compagnie ;

— les objets du niveau supérieur regroupent les données des compagnies en compétition.

La figure 2.3 illustre un exemple d'organisation de données dans laquelle neuf objets de quatre compagnies différentes nommées A, B, C et D sont gérés. Deux classes de conflit d'intérêts montrent les conflits entre A et B d'une part et entre B et C d'autre part. Étant

donné l'organisation ci-contre définie, le modèle de la muraille de Chine restreint l'accès selon les deux propriétés suivantes (Brewer and Nash, 1989) :

— propriété étoile : l'accès en écriture est permis seulement si :
  — l'accès est permis par la règle de sécurité simple, et
  — aucun objet ne peut être accédé en lecture s'il appartient à un ensemble de données de compagnies différentes de celui pour lequel l'accès en écriture est demandé ;

— règle de la sécurité simple : l'accès à un objet o est accordé à un sujet s seulement si l'objet o appartient au même ensemble de données de la compagnie "à l'intérieur de la muraille", ou à une classe de conflit d'intérêts complètement différente.

Le modèle de la muraille de Chine représente un bon exemple des contraintes de la séparation des tâches existant dans le monde réel. Il présente cependant des lacunes sur le fait que l'application stricte des propriétés pourrait se révéler trop rigide comme dans le cas des contrôles d'accès mandataires. Aussi, un autre inconvénient est la sauvegarde et la sollicitation de l'historique des accès qui peuvent à la longue bloquer le fonctionnement du système tout entier. Cet inconvénient devient plus criant dans les environnements collaboratifs dont les accès évoluent de manière exponentielle et la fluidité des accès y est un élément essentiel pour une meilleure collaboration.

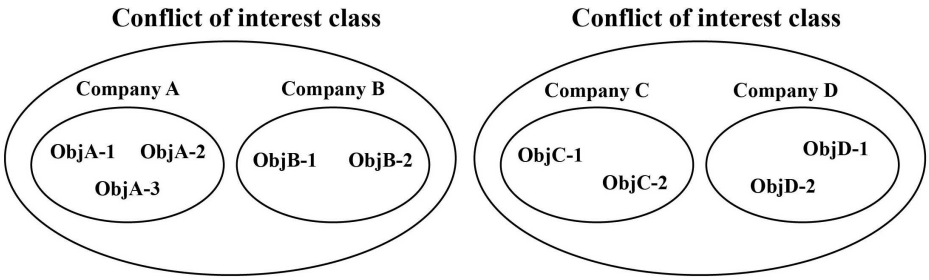


Figure 2.3 Exemple d'organisation d'objets

2.2 Les modèles de contrôle d'accès dans les systèmes collaboratifs

Dans la littérature, beaucoup de modèles de contrôle d'accès ont été proposés. Dans cette section, nous passons en revue les plus significatifs d'entre eux.

2.2.1 Le modèle matrice d'accès

Proposé d'abord par Lampson (Lampson, 1974) pour la protection des ressources dans un contexte de système d'exploitation, le modèle a été ensuite formalisé par Harrison et al. dans (Harrison et al., 1976). C'est un modèle conceptuel qui spécifie les droits que chaque sujet possède sur chaque objet. Le modèle définit trois types d'entités qui sont : les objets

protégés (ressources pouvant être accédées), les sujets (les entités actives qui accèdent aux objets) et les droits d'accès qui associent les sujets aux objets protégés en spécifiant les opérations que les sujets peuvent effectuer sur les objets. La figure 2.4 montre un exemple du modèle de matrice d'accès. Le modèle DAC est le plus souvent limité à l'accès

	File 1	File 2	File 3	File 4
John	Own R W		Own R W	
Alice	R	Own R W	W	R
Bob	R W	R		Own R W

Figure 2.4 Matrice d'accès

d'un utilisateur à un fichier. Seuls les utilisateurs spécifiés dans la matrice d'accès par le propriétaire du fichier ont des combinaisons de permissions en lecture, écriture, exécution ou autres sur le fichier. Il est beaucoup plus utilisé dans les secteurs commerciaux et gouvernementaux. Même si elle représente une bonne conceptualisation des permissions, la matrice d'accès n'est pas appropriée pour une implémentation. Pour un système général, la matrice d'accès sera de très grande taille et creuse qui implique que le stockage de la matrice sous forme de tableau à deux dimensions consomme beaucoup d'espace mémoire. Il existe trois approches pratiques d'implémentation de la matrice d'accès :

— Table de permissions : il s'agit de reporter dans une table à trois colonnes correspondant aux sujets, actions et objets les entrées non vides de la matrice. Chaque tuple dans la table correspond à une autorisation. Cette approche est généralement utilisée dans les systèmes de gestion de bases de données.

— Liste de contrôle d'accès (ACL) (Lampson, 1974) : la matrice est stockée par colonne. Chaque objet est associé à une liste indiquant, pour chaque sujet, les actions que le sujet peut effectuer sur l'objet. Dans la figure 2.5, John a les permissions de lecture et d'écriture sur l'objet "file 1" et il en est propriétaire tandis qu'Alice n'a que le droit de lecture sur "file 1". Dans cette représentation, il est difficile de déterminer tous les accès qu'un sujet a, car cela revient à parcourir la liste de contrôle d'accès de chacun des objets. Il en est de même lorsqu'on doit révoquer les droits d'accès

d'un sujet.

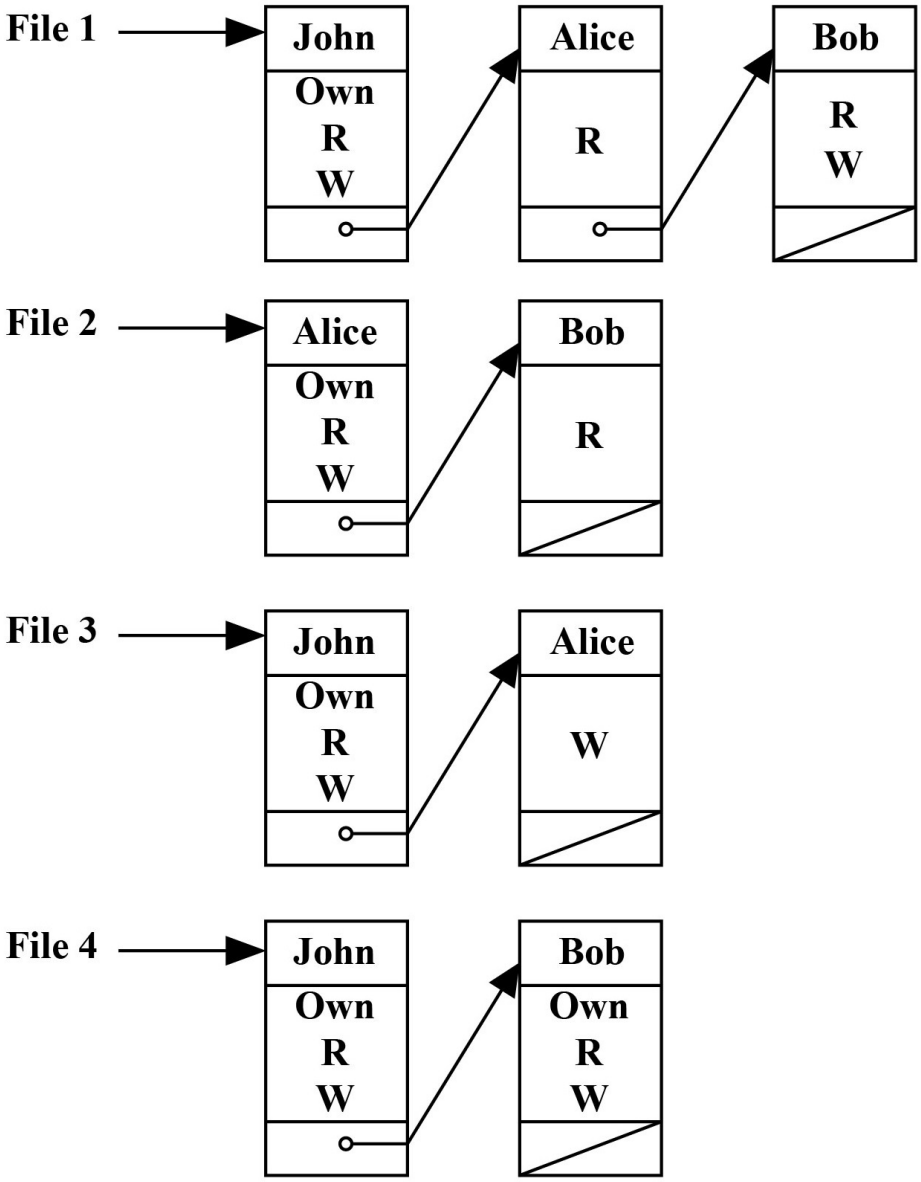


Figure 2.5 Liste de contrôle d'accès

— Liste d'aptitude (C-List) (Lampson, 1974) : la matrice est stockée par ligne. Chaque utilisateur est associé à une liste indiquant, pour chaque objet, les accès que l'utilisateur a droit d'effectuer sur l'objet. Comme l'indique la figure 2.6, le sujet John est associé aux mêmes droits de propriétaire, de lecture et d'écriture sur les deux fichiers "file 1" et "file 2". Il en est de même pour la matrice d'accès associée à Alice et Bob. Comme pour la liste de contrôle d'accès, dans la représentation par la liste d'aptitude, il est difficile de déterminer tous les sujets qui sont autorisés à accéder à un objet particulier.

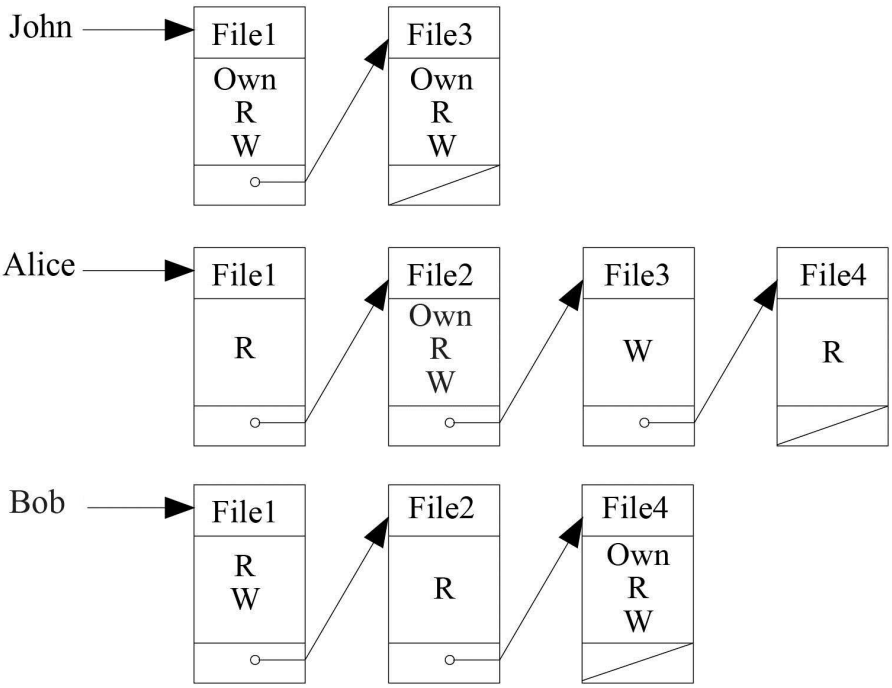


Figure 2.6 Liste d'aptitudes

Le modèle de matrice d'accès a plusieurs faiblesses énoncées non exhaustivement ci-dessous :

— les approches ACL et C-List manquent d'habileté pour supporter les changements dynamiques des droits d'accès comme le changement de responsabilité des utilisateurs.

— Il n'y a aucune restriction sur l'utilisation de l'information quand l'utilisateur l'a reçue ;

— Les privilèges pour l'accès aux objets sont décidés par le propriétaire de l'objet au lieu d'une politique d'ensemble du système qui reflète les exigences de sécurité de l'organisation.

— Ce type de contrôle d'accès est vulnérable aux processus exécutant des programmes malicieux qui exploitent les autorisations de l'utilisateur pour qui ils s'exécutent. Le système de contrôle d'accès peut être outrepassé par des chevaux de Troie embarqués dans les programmes ;

— Les politiques discrétionnaires n'appliquent aucun contrôle sur le flux de l'information une fois que cette information est acquise par un processus. Ce qui rend possible une fuite d'information venant des processus vers des utilisateurs non autorisés à la lire.

2.2.2 Le modèle de contrôle d'accès basé sur les rôles (RBAC)

Afin de combler les lacunes des modèles mandataires, (Sandhu et al., 1996) ont proposé le modèle RBAC. La politique RBAC est calquée sur la structure organisationnelle de l'entreprise. Les responsabilités et compétences de l'utilisateur sont plus importantes que son nom. Le principe de ce modèle est de définir les rôles comme étant une description des tâches ou fonctions qu'un sujet a le droit d'accomplir au sein de l'entreprise. Chaque rôle ainsi défini regroupe un certain nombre de permissions et lui sont associés des utilisateurs qui héritent de facto des permissions liées à ce dernier comme le montre la figure 2.7 ci-dessous.

Les avantages du modèle sont :

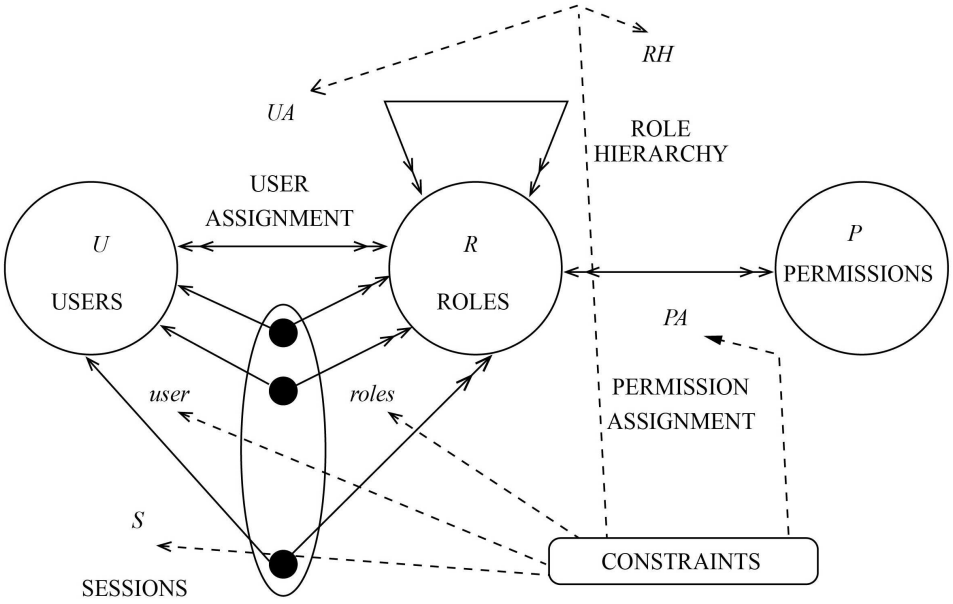


Figure 2.7 Le modèle RBAC

— il est flexible et facile à gérer dans une organisation à grande échelle ;

— les utilisateurs ont le moins de privilège possible, ce qui veut dire qu'on leur octroie le strict minimum dont ils ont besoin pour accomplir leurs tâches ;

— la contrainte de séparation de tâches.

Ce modèle présente néanmoins quelques inconvénients tels que :

— le manque d'habileté à spécifier un contrôle plus granulaire sur des utilisateurs individuels et instances d'objets individuels ;

— l'absence de flexibilité et de réactivité face à l'environnement dans lequel il est utilisé ;

— la difficulté d'obtenir un compromis entre une sécurité forte et une administration simple ;

— le besoin d'une gestion centralisée des affectations utilisateur-rôle et permission-rôle implique que ce modèle n'est pas approprié aux environnements distribués tels que les systèmes collaboratifs. En effet, la gestion devient encore plus difficile quand le sujet et la ressource appartiennent à des domaines de sécurité différents.

2.2.3 Le modèle de contrôle d'accès basé sur les tâches (TBAC)

Proposé par Thomas et Sandhu (Thomas and Sandhu, 1998), le modèle TBAC est une extension du modèle traditionnel sujet/objet qui inclut les domaines qui contiennent de l'information contextuelle basée sur les tâches à accomplir. La figure 2.8 ci-dessous donne un aperçu de l'architecture du modèle. L'accès dans ce modèle est accordé suivant les

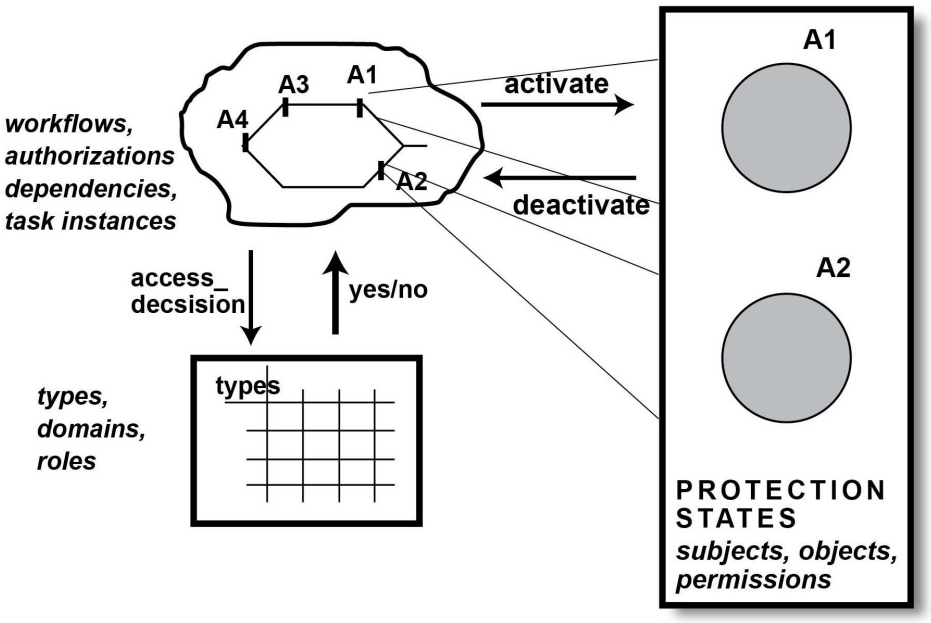


Figure 2.8 Le modèle TBAC

étapes relatives à la progression des tâches liées à un état de protection contenant un ensemble de permissions. Ce modèle est par conséquent qualifié d'actif, car il permet une gestion dynamique des permissions au fur et à mesure de l'exécution de la tâche jusqu'à sa fin. Le modèle TBAC comporte pourtant un certain nombre de lacunes, parmi lesquelles nous pouvons citer :

— il se limite aux contextes en relation avec les activités, les tâches et la progression du flux de travail. Ce qui est insuffisant pour les systèmes collaboratifs qui requièrent une définition plus large de la notion de contexte et sont souvent difficiles à partitionner en tâches ;

— il peut introduire un trop grand nombre de contraintes à travers les flux de travail s'il est géré par un module de contrôle d'accès centralisé.

— Il est naturellement plus adapté à l'entreprise dont la caractéristique principale est la division du travail en tâches.

2.2.4 Le modèle de contrôle d'accès basé sur les équipes (TMAC)

Ce modèle proposé par (Thomas, 1997) définit deux aspects importants du contexte de collaboration : le contexte d'utilisateur et le contexte d'objet. Le contexte d'utilisateur donne un moyen d'identifier des utilisateurs particuliers jouant un rôle dans un groupe à un moment donné. Le contexte d'objet identifie des objets spécifiques requis pour la collaboration à ce même moment. Il offre la possibilité de spécifier des contrôles fins sur des utilisateurs individuels dans certains rôles et sur des instances d'objets individuels. La figure 2.9 donne un aperçu de ce modèle. Toutefois, le modèle TMAC manque d'auto-

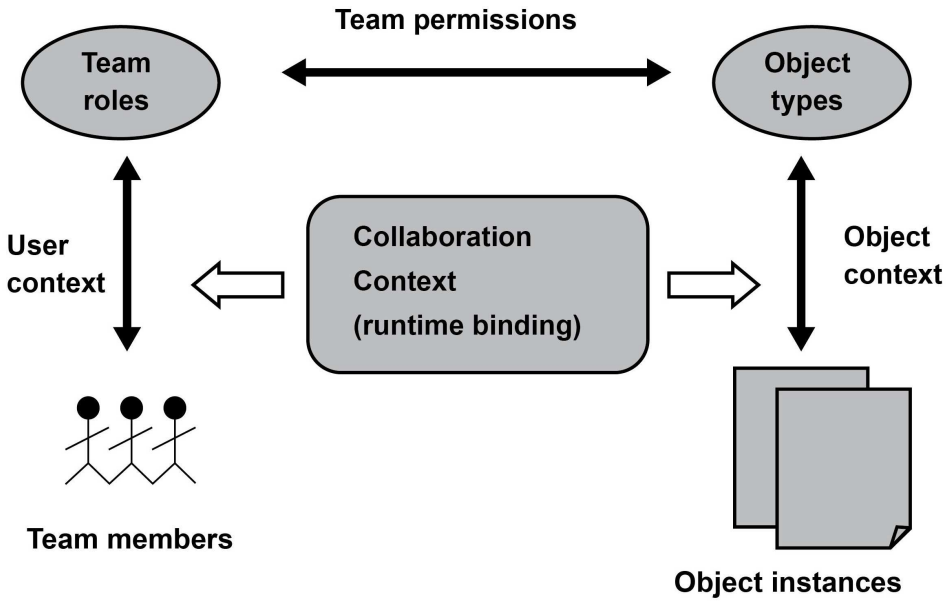


Figure 2.9 Aperçu du modèle TMAC

administration des relations d'affectation entre les entités qui plus est, la spécification, la sélection et l'application des politiques multi paramètres requièrent de plus amples analyses. (Georgiadis et al., 2001) ont proposé l'intégration des modèles TMAC et de RBAC en incorporant la notion de contexte comme une entité dans l'architecture. Ce modèle est le C-TMAC (TMAC contextuel) et vise à inclure l'information contextuelle dans la prise de décision du contrôle d'accès.

2.2.5 Le modèle du contrôle d'accès basé sur les groupes et les tâches (TT-RBAC)

Proposé par (Zhou and Meinel, 2007), le modèle TT-RBAC est une extension du modèle NIST RBAC (Ferraiolo et al., 2001) intégrant la notion de tâche et d'équipe. Comme

l'indique la figure 2.10 ci-dessous, pour l'assignation des permissions, les utilisateurs individuels sont affectés aux équipes, les rôles et les tâches sont affectés aux équipes et les permissions sont, quant à elles, assignées aux tâches. Des relations à cardinalité de n à n sont ainsi définies entre les entités. Encore une fois, ce modèle est beaucoup plus adapté à la configuration d'une entreprise où la définition des tâches et des groupes de sujets est beaucoup plus aisée comparativement aux systèmes collaboratifs ouverts.

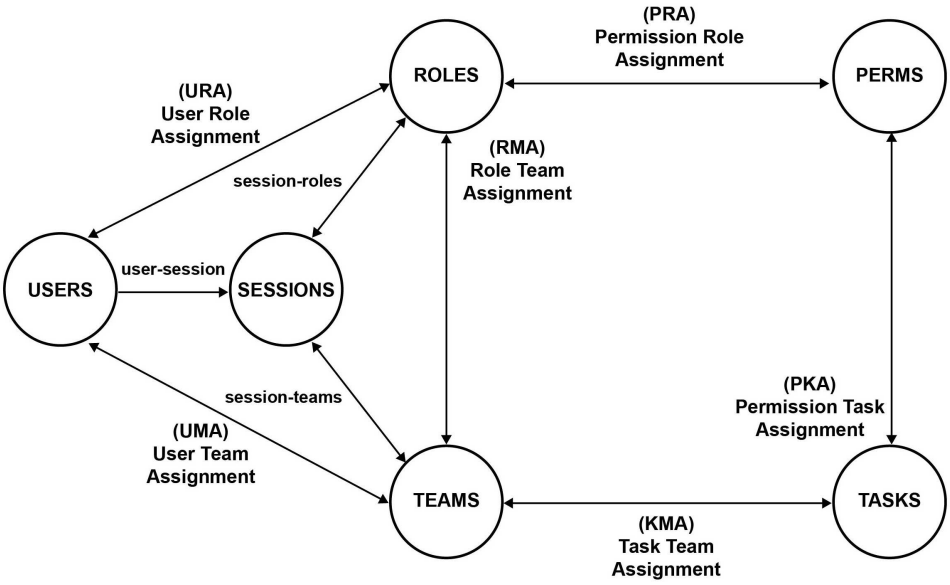


Figure 2.10 Architecture du modèle TT-RBAC

2.2.6 Le modèle de contrôle d'accès basé sur le contexte (CBAC)

Ce modèle proposé par (Covington et al., 2001) est une extension du modèle RBAC avec la prise en compte de la notion de rôles d'environnement dans le but de fournir plus de sécurité aux applications sensibles au contexte. Le modèle permet de cerner le contexte de sécurité de l'environnement dans lequel les requêtes d'accès sont faites. En effet, les rôles sont activés sur la base des conditions d'environnement au moment où les demandes sont effectuées. Même s'il capte le contexte dans lequel les requêtes d'accès sont effectuées, ce modèle manque de cas d'application permettant de démontrer sa robustesse.

2.2.7 Le modèle de contrôle d'accès basé sur les attributs (ABAC)

Au lieu de seulement considérer les sujets, les objets et les contextes, le modèle ABAC qui a été proposé par (Kolter et al., 2007) introduit la notion d'attribut relatif aux différentes entités impliquées dans la collaboration afin de prendre la décision définitive pour le contrôle d'accès. La figure 2.11 ci-dessous donne une vue du modèle ABAC dont les principaux concepts sont :

— le concept d'attribut : c'est une variable qui capte les caractéristiques de l'entité ;

— le concept de sujet et d'attribut de sujet : un sujet est une entité qui agit au nom d'un utilisateur. Les attributs d'un sujet indiquent les propriétés du sujet telles que le nom du sujet, ses rôles, les groupes auxquels il appartient, les tâches qui lui sont assignées ou encore sa réputation ;

— le concept d'objet et d'attributs d'objet : un objet n'est rien d'autre qu'une instance d'une ressource protégée. Les attributs d'objet sont quant à eux des propriétés de l'objet telles que son utilisation, son type, sa localisation, sa version, etc. ;

— le concept de contexte : regroupe les circonstances dans lesquelles l'accès se produit et sont représentées par un ensemble de paramètres opérationnels et techniques ;

— le concept d'opération : il représente les actions qui peuvent être effectuées sur les objets comme par exemple créer, lire, mettre à jour, et supprimer ;

— le concept de permission : c'est l'ensemble des autorisations permettant d'effectuer une opération sur des ressources, la combinaison d'un sujet et d'une opération sur un objet ;

— le concept de fonction d'affectation de permissions : il définit les fonctions utilisées pour appliquer les politiques de contrôle d'accès et les règles à une requête de contrôle d'accès.

Le modèle ABAC présente plusieurs avantages dont les principaux sont :

— il est intuitif quand il s'agit de modéliser et de gérer les politiques de contrôle d'accès du monde réel ;

— il est plus flexible et plus puissant pour la représentation des sémantiques des contrôles d'accès granulaires qui sont spécifiquement appropriés pour les systèmes collaboratifs ;

— la gestion de l'information liée à la sécurité est repartie sur un certain nombre d'attributs et de pouvoirs des politiques souvent au-delà des frontières organisationnelles. Ce qui est approprié pour le partage de l'information à grande échelle ;

— il réduit la complexité globale du système en permettant aux différents composants du système de se focaliser sur leurs tâches administratives respectives.

Malgré son énorme potentiel, le modèle ABAC présente une limitation majeure liée au nombre d'attributs impliqués dans la définition de la politique de contrôle d'accès. En effet, le nombre de rôles et permissions requis pour encoder ces attributs va croître de manière exponentielle de sorte que les affectations des utilisateurs et celles des permissions deviennent difficiles à gérer.

2.3 Codage collaboratif

De nos jours, le développement de logiciels de grands systèmes est un processus hautement collaboratif dans lequel le travail de groupe est essentiel. À travers ce processus, les

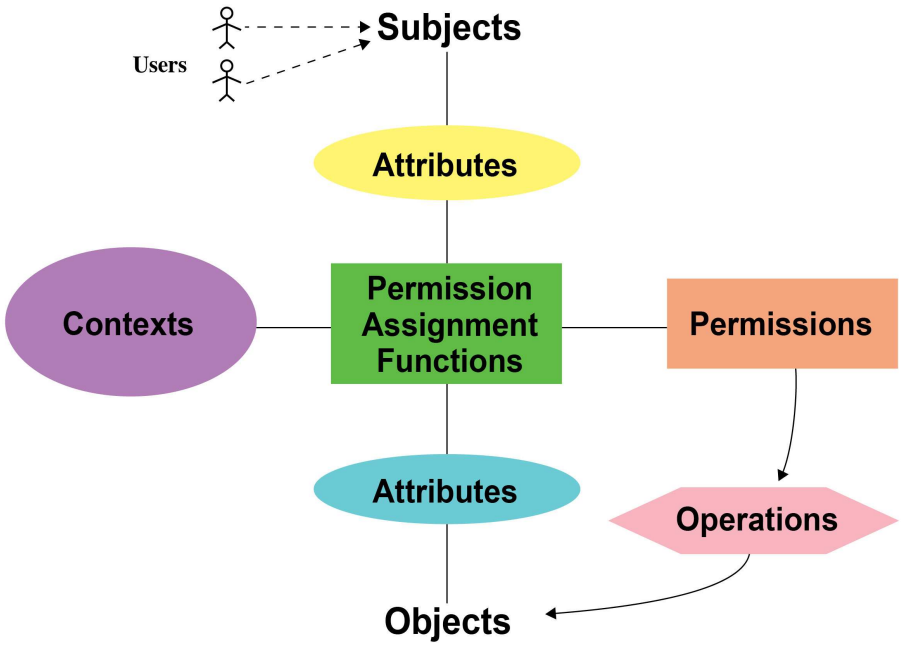


Figure 2.11 Le modèle ABAC

programmeurs travaillent ensemble sur un même code en utilisant exclusivement l'une des deux stratégies : éditer une copie unique du code en se basant sur un IDE ou éditer des copies distinctes en parallèle en intégrant leurs efforts à travers un système de contrôle de version de code source.

Dans la littérature on distingue deux grands groupes de méthodologies dans le domaine du codage collaboratif soient, les systèmes de codage collaboratif non temps réel et temps réel.

2.3.1 Codage collaboratif non temps réel

Dans cette catégorie, Pilato propose le contrôle de version (Collins-Sussman, 2002) tandis que Berliner propose le système de version concurrente (Berliner et al., 1990). Dans ces systèmes, les programmeurs individuels éditent le code source dans leur espace de travail local et synchronisent manuellement leurs mises à jour.

2.3.2 Codage collaboratif temps réel

Dans ce type de codage collaboratif, plusieurs programmeurs éditent le même code source partagé au même moment. À l'heure actuelle, les principales méthodologies de programmation proposées à date dans le volet collaboratif sont la programmation par paire (pair programming en anglais) et la programmation côte à côte (side-by-side programming en anglais),

Programmation par paire

La programmation par paire est une pratique permettant à deux programmeurs de travailler sur le même code source dans un environnement physique unique de développement. Les objectifs de la programmation par paire sont :

- améliorer la communication et le partage de la connaissance dans un groupe ;
- augmenter la productivité ;
- améliorer le processus du développement et aider les membres de l'équipe de développement à demeurer productifs ;
- et finalement, renforcer la qualité du logiciel résultant (Beck and Gamma, 2000).

Dans (Dawande et al., 2008), les auteurs citent un certain nombre d'études selon lesquelles la programmation par paire réduit l'effort du programmeur, les erreurs logicielles et la complexité du code. (Bryant et al., 2008) ont, dans leur article, fait la comparaison entre les performances de la programmation par paire, la programmation en solo et la combinaison des deux méthodes en considérant deux objectifs : la minimisation de l'effort et la minimisation du temps de programmation. Ils sont parvenus à la conclusion que la programmation par paire est préférable quand il s'agit d'un partage efficace de connaissances entre programmeurs ou d'une conception de systèmes fortement connectés. Cette méthode est également la plus efficace pour améliorer l'expertise en associant les experts aux partenaires moins qualifiés. La programmation en solo est préférable dans le cas d'un système large où l'effort nécessaire pour former une paire est trop élevé. Quant à l'approche mixte, elle est indiquée dans un système composé de modules qui sont suffisamment différents les uns des autres.

La programmation côte à côte (side-by-side programming)

(Cockburn, 2004) propose la programmation côte à côte dans laquelle "deux personnes s'assoient assez proches pour voir facilement l'écran de l'un et de l'autre, mais chacun travaillant sur les tâches qui lui sont assignées". (Prechelt et al., 2008) ont discuté des différents objectifs qui se cachent derrière les épisodes collaboratifs entre les programmeurs de la méthode côte à côte. (Dewan et al., 2009) dans leur article, ont trouvé qu'il était préférable d'avoir la programmation côte à côte à distance permettant aux programmeurs de collaborer dont l'un ou les deux éditant le code au lieu d'en avoir un seul qui édite et l'autre qui suit son partenaire.

Dans le cadre du codage collaboratif, plusieurs études ont été faites pour implanter des systèmes d'édition collaborative permettant de doter les programmeurs d'outils flexibles et performants favorisant leur regroupement pour travailler sur le même projet de développement de logiciels afin de gagner en temps, en expérience et en efficacité.

2.3.3 Outils de développement collaboratif

Il existe une panoplie d'outils proposés par les éditeurs commerciaux ainsi que par les éditeurs de logiciels libres :

— composants d'édition collaborative : *MobWrite*[1], *EtherPad*[2] ;

— environnements de développement intégré collaboratifs : *StudioPad*[3] utilise *EtherPad* pour la programmation collaborative graphique. L'IDE *Cloud9*[4] inclut la collaboration temps réel dans leur environnement de développement web. Google a livré ses travaux sur *Collide*[5] qui est un IDE avec édition collaborative temps réel basé sur le protocole *Wave*[6].

Les travaux de recherche sur les environnements de développement collaboratif incluent *Adinda* (van Deursen et al., 2010), un éditeur web adossé à Eclipse et *CoRED* (Collaborative Real-time EDitor) (Lautamäki et al., 2012), un IDE Java pour le développement d'applications Web utilisant la notion de synchronisation différentielle. Développée par (Fraser, 2009), elle rend possible l'édition collaborative : une technique dans laquelle les clients et les serveurs échangent les différences au lieu des opérations de modification. Ce qui peut entraîner des conflits d'édition quand plusieurs éditeurs manipulent la même partie du texte. *CoRED* ne résout pas le problème d'interférence entre un programmeur et un collaborateur, il utilise des notes qui permettent aux programmeurs de laisser des messages attachés à des zones dans le code source et des verrous quand un programmeur peut obtenir des droits exclusifs pour modifier une partie du code. L'outil *CollabVS* est proposé par (Hegde and Dewan, 2008) pour permettre une collaboration ad hoc afin de partager les mêmes fonctionnalités de collaboration et de sensibilisation. Plusieurs études dans la littérature ont fait écho à la notion de sensibilisation (awareness en anglais) dans les outils de codage collaboratif. Parmi ces travaux on peut citer : *SAROS* (Salinger et al., 2010), un plug-in pour une séance de programmation distribuée se focalisant sur la sensibilisation dans la collaboration et permettant la création d'autant de groupes virtuels que de besoins. L'outil *Collabode* proposé par (Goldman et al., 2011) est un environnement de développement intégré web conçu pour expérimenter la manière dont deux programmeurs peuvent écrire un code en étroite collaboration l'un avec l'autre. *Palantir* (Sarma et al., 2003) est un outil d'espace de travail collaboratif et de sensibilisation qui vient en complément aux systèmes de gestion de configuration existants en fournissant aux programmeurs des vues sur les autres espaces de travail. Il se focalise sur la collection, l'organisation et la présentation d'information pertinente sur les espaces de travail. *Syde* (Hattori and Lanza, 2010) est un plug-in Eclipse qui enregistre toutes les modifications

1. `code.google.com/p/google-mobwrite`
2. `etherpad.org`
3. `sketchpas.cc`
4. `cloud9ide.com`
5. `code.google.com/p/collide`
6. `waveprotocol.org`

effectuées dans les projets multiprogrammeurs et les diffuse en temps réel à tous les programmeurs travaillant sur le même système. (Lanza et al., 2010) ont combiné *Scamp* un autre plug-in Eclipse (faisant de la sensibilisation des espaces de travail) et Syde afin d'accroître le niveau de connaissance des programmeurs sur ce qui se passe dans le projet en termes de modifications dans l'implantation.

D'autres études se sont focalisées sur l'analyse de l'impact des modifications sur le reste du code. Parmi celles-ci, (Ryder and Tip, 2001) ont proposé une méthode consistant à faire une correspondance entre les modifications du code source et un ensemble de modifications atomiques précédemment défini. *Chianti*, proposé par (Ren et al., 2004), est un outil d'analyse d'impact pour Java qui utilise deux versions d'une application et décompose leurs différences en un ensemble de modifications atomiques. (Wloka et al., 2009) proposent un outil d'analyse de la validation sécurisée des modifications d'un code source qui permet de déterminer quelles modifications du code peuvent être validées de façon sécurisée dans le but du contrôle de version. (Shao et al., 2007) utilisent les dépendances de données et les découpages en tranches des programmes pour détecter l'interférence sémantique entre les modifications faites en parallèle. Plus récemment, (Fan and Sun, 2012a) ont développé *CoEclipse* qui convertit l'éditeur de code source Eclipse mono-utilisateur en un outil de programmation collaborative multi-utilisateur. Il intègre la cohérence syntaxique à travers la transformée opérationnelle (Sun and Ellis, 1998), et la cohérence sémantique à travers le blocage automatique basé sur les dépendances. (Bravo et al., 2007) ont proposé *COLLECE* pour l'apprentissage collaboratif. Il supporte l'édition collaborative, la compilation et l'exécution des programmes dans un mode distribué et synchrone et il inclut des outils avancés de communication, de coordination, et de sensibilisation sur les espaces de travail. Pour terminer, *RECIPE* (Real-time Collaborative Interactive Programming Environment) proposé par (Shen and Sun, 2000) est un autre environnement de codage collaboratif qui permet à des programmeurs géographiquement distribués de participer de manière concurrente à la conception, au codage, au test, au débogage et à la documentation d'un programme.

Les systèmes de codage collaboratif ci-dessus cités peuvent être regroupés selon :

- architecture : client-serveur ou P2P,
- conception de données : centralisées ou répliquées,
- technique de maintenance de la convergence : certaines approches favorisent la résolution de conflits quand ils surviennent, et d'autres assurent la prévention de conflits par l'utilisation de la technique de la transformée opérationnelle.

Nos travaux de recherche sont uniquement basés sur une architecture P2P avec des données répliquées et utilisant une technique d'OT pour assurer la convergence. Dans cette catégorie, les solutions proposées dans la littérature utilisent le plus souvent l'algorithme de transformation inclusive de (Sun and Ellis, 1998) qui malheureusement n'assure pas la convergence dans ce contexte-là selon les recherches de (Randolph et al., 2015).

CHAPITRE 3
ARTICLE 1 : FORMAL PROOF OF WISESHARE : A COLLABORATIVE ENVIRONMENT FOR KNOWLEDGE SHARING GOVERNED BY ABAC POLICIES

Moustapha Bande, Hakima Ould-Slimane, Hanifa Boucheneb
Soumis pour publication au journal *Information Security Journal : A Global Perspective.*

Abstract - In this paper, we propose an attribute based access control (ABAC) approach for safely sharing knowledge in a collaborative environment. Indeed, existing similar systems facilitate collaboration at the risk to convey doubtful information and sometimes serve as a gate to vandalism. Our system called "WiseShare" ensures collaboration while focusing on the reliability of the broadcasted content. To achieve this goal, we precisely express the requirements needed to control the shared information. In addition, we define a formal framework for specifying security policies governing contributions requests according to user profile. We present also a prototype implementing the functionalities of our system. In our approach, a new user has limited rights. On an ongoing basis, if he demonstrates his ability to produce good contributions, he gains new privileges. Conversely, if he often generates contributions with a questionable content, he loses the held privileges. Therefore, granting user requests depends closely on his previous behavior (the history). Indeed, our system encourages users to adopt a responsible attitude by focusing on the reliability of the content of their contributions instead of their frequency. Consequently, WiseShare permits to significantly minimize clumsy and malicious actions.

Keywords - Collaborative environments ; Web 2.0 platforms ; knowledge sharing ; attribute based access control ; model checking.

3.1 Introduction

During the last decades, the need of sharing information induces the large proliferation of collaborative environments. Indeed, using this computing fashion results in substantial gain both in the output quality and in the reduction of calculation time. These collaborative systems are widely and successfully used by shared computation systems and collaborative edition platforms. Moreover, with the emergence of systems based Web 2.0, the collaboration concept becomes essential for the viability of many popular applications such as Wikipedia, Flickr, Youtube, Del.icio.us, etc. However, the large sharing and manipulation of information between users in a collaborative environment impose a tight monitoring on their activities. Usually, to ensure trustworthiness of the shared resources, access control policies must be defined and enforced during the collaboration. In the literature, there are many access control models (Sandhu et al., 1996), (Thomas and Sandhu,

1998), (Thomas, 1997), (Kolter et al., 2007) each of them focuses on a particular dimension related to users, resources or relation between them. Generally, users are seeking for flexible systems that allow them to easily share information. However, ensuring flexibility and efficiency of the collaborative systems while controlling the access to the involved resources is a great challenge.

Although, access control models in collaborative systems are various and heterogeneous, there are some common and specific requirements that can be defined for their evaluation. In (Jaeger and Prakash, 1996), the authors present basic requirements that any access control model in a collaborative environment should meet. Thus, an effective access control model :

- should be generic and expressive ;
- requires great scalability ;
- must be able to protect information and resources of any type and at varying level granularity ;
- must facilitate transparent information access to authorized users and strong exclusion of unauthorized ones in a flexible manner ;
- must allow high-level specification of access rights ;
- must be dynamic (specify and change policies at runtime) ;
- should keep performance and resource costs within acceptable bounds.

Driven by the public passion for online information search, we investigated the famous collaborative encyclopedia "Wikipedia". After a critical review, we could enjoy the usability and the richness of the encyclopedia. However, from a security point of view concerning the management of the contribution content, we identified some drawbacks mostly due to the high accessibility that the platform offers to users. In fact, this popularization enables an immediate dissemination of each contribution independently of its content, and thus, is conducive to inappropriate and malicious activities of users. To overcome this hyper-permissiveness, we propose in this paper a collaborative system that we called "WiseShare" for creating and sharing knowledge while security is being closely ensured. More precisely, we aim to control the collaborators activities so that only reliable contributions are broadcasted. For our design purposes, we need an access control model that is capable to define fine-grained level permissions and offers the possibility to handle dynamic changes on our collaborative environment. Consequently, in our system, users may have different permissions for their contributions. These permissions are dynamically and automatically generated by the system which is based on the profile and the behavior of users through their contributions. The system may update the profile of a user each time he submits a contribution. It is clear that the definition of static roles cannot deal with our system requirements. Thus, the ideal candidate for our specific design purposes, is the ABAC model (Kolter et al., 2007). Indeed, ABAC proposes the definition of permis-

sions based on any security-relevant characteristics (the attributes). This model is suitable for our approach since we are planning to control the collaboration system through the contributor's attributes, the contribution attributes, as well as the context of the collaboration. The rest of this paper is organized as follows. Section 3.2 discusses the related work concerning access control models for collaborative systems. Section 3.3 focuses on social based collaborative environments and provides a critical overview of "Wikipedia". Section 3.4 describes the foundations of our approach, the requirements and the targeted objectives. Section 3.5 provides a formal framework for specifying the components of our approach (user, contribution and context) as well as the ABAC policies reflecting our requirements. Section 3.6 presents the algorithms depicting the impact of performing contributions on users profile according to the defined ABAC policies. Section 3.7 depicts a prototype implementing our system (WiseShare). Finally, Section 3.9 provides some concluding remarks and future work.

3.2 Related Work

In the literature, several access control models have been proposed for collaborative systems. Role-based access control (RBAC) which has been proposed by (Sandhu et al., 1996) is based on the organizational structure of the enterprise and maps permissions to roles and roles to users. Even though it is flexible and easy to manage, RBAC is not suitable for all collaborative systems since it cannot handle fine-grained control on individual users and doesn't include the impact of context in taking access control decision. To take into consideration contextual information, Task-based access control model (TBAC) (Thomas and Sandhu, 1998) has been introduced as an extension of RBAC. TBAC introduces dynamic management of permissions during the progress of the task until its completion. While considering collaborative system access control, TBAC fails to be more efficient since it has to manage activation and deactivation of permissions during task execution. Team-based access control (TMAC) (Thomas, 1997) and Contextual TMAC (C-TMAC) (Georgiadis et al., 2001) are other extensions of RBAC and offer the possibility to specify fine-grained access control. They define context on user level and context on object level. However, TMAC and C-TMAC lack to clearly define the assignment of relations between entities. Team and Task-based RBAC (TT-RBAC) (Zhou and Meinel, 2007) adds the concepts of team and task into RBAC model while ensuring context-aware access control. In this model, users are assigned to teams, tasks and roles to permissions and permissions to tasks. Context-based access control (CBAC) (Covington et al., 2001) is another extension of RBAC by considering environment roles. In this model, roles are activated based on environment conditions at the time of the request. However, all these proposed models have some shortcomings while considering collaborative environments since they are either incomplete or addressed a specific need and subject to insufficient insurance. Instead of only considering subjects, objects and contexts, attribute-based access control

(ABAC) has been proposed in(Kolter et al., 2007) ; it introduces the concept of attribute related to the different entities involved in collaboration to make the access control decision. (Smari et al., 2009) proposed a scheme to incorporate trust and privacy in the ABAC model. In this scheme, the level of trust is affected by context and subject attributes and has an influence on the access control decision. Similarly, privacy preserving is also measured by ensuring that a requested object is used accordingly to the access purpose of the subject. In order to allow a large number of users to gain benefit of sharing, the authors in (Wang et al., 2009) proposed a novel ABAC framework based on people tagging. The main idea is to identify user attributes from the information provided by the collaborative efforts of the system users. The authors also propose a formal language for tag-based policy specification. This model may be vulnerable to malicious users' collision since one can gain faked tags from partner users. (Nasirifard et al., 2008) proposed an annotation-based access control model for collaborative and social platforms. The model also uses the concept of "tagging" for annotating shared resources and is applicable in multiple Web-based collaboration systems. However it lacks to take into account contextual information. (Demchenko et al., 2006) proposed a model using ABAC for securing web services and Services Oriented Architectures (SOA). They discuss also a detailed comparison between RBAC and ABAC models.

3.3 Social Collaborative Environments

The social computing (web 2.0) implies any form of technological progress that makes the web more simpler, more accessible and mainly more ergonomic. This innovative environment allows users from different horizons to take full advantage of the web functionalities as well as the online conveyed content. Moreover, this revolution turns the common passive user checking only information into an active actor. Hence, the users have voluntary responsibilities in content creation and sharing. In fact, large web collaboration allows popular application blossoming such as Wikipedia (the collaborative encyclopedia) and Del.icio.us (bookmarks sharing tool). Users have the choice and the power to collaborate by bringing a contribution, as little it can be. Thus, every user of the collaborative system has the right to benefit from the whole information even though he did not contribute. Faithful to the web 2.0 philosophy, these systems offer a large flexibility for broadcasting any contribution (regardless of its source) which may be subject to the continuous updates of users. In fact, the control is ensured in a collaborative and voluntary way. In other words, the main purpose of these systems is to encourage sharing and collaboration instead of their restriction. From a security point of view, it represents a big potential of hidden vulnerabilities in this hyper-permissive collaboration. In order to better illustrate our opinion, we have investigated the free encyclopedia Wikipedia. Actually, this online encyclopedia is one of the most visited search engines on the web. Most people are using it as the first, even the single reliable reference. The information broadcasted by this ency-

clopedia is collected in a collaborative way. Moreover, Wikipedia's catchword is : "everyone is welcome to contribute". Fortunately, Wikipedia is designed mostly by the collaborative efforts of honest contributors who share voluntarily their knowledge. However, it would be careless to give absolute confidence to all the content disseminated in Wikipedia. Indeed, it is impossible to ensure either the honesty or the competence of all the contributors being active at any time on the system. At the organizational level, Wikipedia's volunteers are classified by their status. Therefore, the underlying access control model is the RBAC model. It mostly contains the following roles : administrators, bureaucrats, IP address inspectors and the bots (spell-checkers). In Wikipedia, any registered user has the right and the privilege to immediately disseminate his contribution regardless of its content and without any verification. Therefore, a contribution with an inaccurate or malicious content may be visible until someone intervenes to modify or remove this contribution. Moreover, the expertise notion does not exist for the contributors ; there are only thematic portals for classifying the contributions. Any user can also modify a contribution posted by another one, mostly even though he is not skilled to do it. He can also disseminate his modifications without being authenticated. In this case, the IP address of such a user is publicly revealed. To prevent inappropriate modifications or vandalism actions, a history of all the modifications is kept. This history is used to undo an inadequate modification in order to get back to a previous correct version. Nevertheless, it is sometimes difficult to retrace the history to reach the original version. In addition, the prevention of inappropriate and malicious actions is ensured by a category of selected people (administrators and bureaucrats). So, their presence in the encyclopedia system is required all the time since they are also responsible for blocking users (who behave as vandals) as well as promoting them to privileged rank. Therefore, every administration intervention is done manually based only on the volunteers' decision. Consequently, the collaboration is widely facilitated compared to the reliability of the disseminated information. This is the issue that led us to think about balancing the goal of flexible collaboration and reliable contribution. Mainly, we aim to propose an automatic mechanism for accessing and controlling user's profile trough their activities (contributions). These profiles are implemented according to the ABAC model. Furthermore, each contribution request must respond to a specific security policy and may potentially update its owner profile.

3.4 WiseShare : Description of Our Approach

In this section, we describe "WiseShare" our ABAC based collaborative system for knowledge sharing. The main objective of this system is to allow a reliable sharing of knowledge where security is strongly enforced. Our system consists of three main components : the contributors (collaborators, users), the contributions (the knowledge being shared) and the environment (the context) under which the collaborative contributions occur.

The overall philosophy of our system is based on the following features :

- Do not give all users the same rights of contribution : indeed, it would be wise to grant the contribution rights according to the user profile. So, an expert contributor may not have the same rights as a novice one.
- The access rights based on ABAC model are automatically generated from the contributor's profile. In our system, access control policies are mainly defined according to the profile characteristics of the contributor.
- The contributors are involved as little as possible in the management of the contributions content made by other users (editing tasks are limited to experts). That helps to preserve the enthusiasm of the contributors since they do not have to permanently monitor the new contributions or track the history of contributions.
- The contributor must register. Hence, there is no need to reveal their IP addresses because there is more control on background. Indeed, a safe collaboration requires an authentication to map users to their contributions.
- The global collaborative behaviors of contributors "decide" of their "rank" in the system. Indeed, unlike the work in (Wang et al., 2009), our approach is using a behavior-based tagging which is not based on users'opinion.
- Each request can potentially change the attributes of the system components (which are : users, contributions, environment). So, contributing becomes an activity taking place in a dynamic environment. It evolves according to the global contributions effect.
- Confidence is a volatile resource (no one is an expert forever) and is guaranteed only if the user behavior confirms it. Hence, there is a continuous assessment of the contributor behavior upon each issued contribution .
- We can distinguish three kinds of contributors : novice, expert and vandal. Each new user is noted as novice. If he demonstrates his expertise in a certain topic, he becomes an expert in this topic. However, if he is massively tagged as a malicious contributor, he turns into " vandal " and thus, he is blocked (he loses the right to contribute).
- Each new contributor has to demonstrate his expertise. The user builds his reputation according to his interventions, so he does not have to claim to be an expert. The expertise is proved if the user succeeds to disseminate a certain number of significant contributions in a specific domain.
- Each query (creating, publishing, editing, suppressing, reporting vandalism, etc.) must satisfy an ABAC policy based on the contributor's reputation (automatically generated tag) ;
- when there is detection of suspicious behavior (from a novice or a vandal user), the experts are automatically notified to intervene.
- The ABAC Policies governing the system can include :

- The query binding a user to a contribution, like : creating, editing (modification), publishing, suppression.
- Or the opinion a user can have about another user, like : reporting a vandalism case.

Mainly, the collaborative strategy adopted by WiseShare aims to achieve the following objectives :

- Controlling the contribution activities (and therefore sharing) to prevent vandalism and clumsy contributions.
- Reducing users intervention.
- Automatically profiling contributors according to their global contributory behavior.
- Optimal and automatic management of the history of contributions modifications (ideally suppressing it).
- Speed and automatic prevention and detection of vandalism.
- Encouraging honest contributors to step back by reviewing their contributions (pass it through a spell checker, check their sources...etc.).
- Automating as possible the processing of contributions.
- Updating the status of contributors according to the quality and the frequency of their contributions.

3.4.1 Access Control Requirements In WiseShare

In this section, we present the outlines of our solution. We suppose that a contributor u would like to publish a contribution c in a collaborative environment e controlled by a set of ABAC policies P. The figure 3.1 shows the evolution of the user's profile through the three main states (novice, expert and vandal).

Each contribution request r expressed by a user u has to satisfy the following principles :

- Every new user of the system is tagged "novice" by default.
- A novice user has just the right to create temporary pages which are only visible to the experts in the contribution topic. Hence, a novice contributor is unable to broadcast immediately his contributions to the public.
- The creation of temporary pages automatically notifies the experts for editing the new contribution if it is necessary.
- After one week, if the temporary contribution is neither suppressed nor radically changed, this contribution becomes visible for the public, and is counted (recorded) for the original author (in the concerned topic).

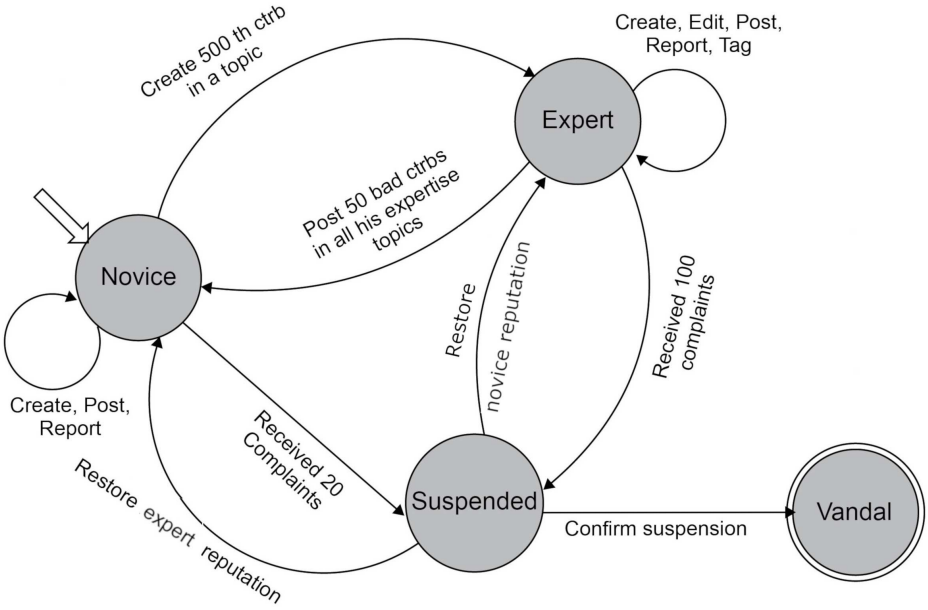


Figure 3.1 Evolution of users'reputation in WiseShare.

- If a novice achieves n (for instance 500) recorded contributions in a certain topic, he will automatically receive the tag "expert" in this topic.
- if an expert modifies drastically the content of a contribution, he becomes the main author of this contribution. This does not mean that the expert becomes the owner of the contribution idea. Indeed, the original author of the contribution never changes.
- If an expert contributor publishes a m (for instance 50) irrelevant contributions (removed or significantly corrected contributions) in his field of expertise, he loses the reputation of expert in this topic. If he is no longer expert in all the topics, he becomes unreliable and receives the tag "novice" and loses all his privileges.
- Every contributor may denounce the contents of a contribution made by a potential vandal. However, he cannot denounce it more than once. In addition, the reporting of a vandalism case has to be justified.
- If a contributor receives k complaints (for instance 20 for a novice user and 100 for an expert user), he is automatically tagged as "Suspended". If an expert accepts these complaints then the owner of the tagged contributions is tagged as "Vandal" and he is banned from the system. Otherwise, he recover his initial reputation.

  Remark : In our system, the displayed contributions are anonymous. Therefore, users tag contributions not contributors. Hence, malicious users can not corrupt the system by collaboratively tagging a honest user as "Vandal".
- A contributor user u can only express contribution requests r among the set {Create, publish, Edit, suppress, report, decide }.
- If the request r satisfies the corresponding policy from the set P, it is granted ;

otherwise, it is denied.

3.5 ABAC Policies Specification

In this section, we present our formal framework for specifying the attribute based access control policies required by WiseShare. We recall that our system consists of three entities : User (contributor), Contribution, Environment. Formally, each component is a tuple of attributes as follows :

- The set of user attributes $\mathcal{U}$: define the identity and the characteristics of a user who contributes in the system. A contributor u is represented by a tuple as follows : $u =< idu, rep, Skl, cb, ncpl >$ where :
  - ▷ idu is a unique string denoting the identity of the user who must register to the system.
  - ▷ rep denotes the reputation of the user, where $rep \in \{$"$novice$", "$expert$", "$permanentexpert$", "$suspended$", "$vandal$"$\}$. We note that the "permanent expert" is a kind of the system administrator.
  - ▷ Skl denotes the set of expertise a contributor has gained during his collaboration in the system (the user skills, for instance : $Skl \subset \{$"Art", "$Sciences$", "$Sport$", "$Philosophy$", ...$\}$). By default, it is empty when he registers.
  - ▷ cb gives for each topic the number of recorded contributions proposed by this user. By assuming k topics, we have : $cb = \{cb.t_1, \ldots, cb.t_k\}$, where t_i is the i^{th} topic.
  - ▷ $ncpl$ denotes the number of the received complaints made by distinct users.
- The set of contribution attributes $\mathcal{C}$: defines the features outlining a contribution. These attributes serve to locate and identify a contribution in the system, so that it cannot be confused with another one. A contribution c is given by a tuple : $c =< idc, d_{cr}, orig, chf, th, vis, Cpl, ctt >$ where :
  - ▷ idc denotes the identifier of the contribution (a string).
  - ▷ d_{cr} denotes the creation date of the contribution in the system.
  - ▷ $orig$ denotes the original author of this contribution (he can be a novice or an expert).
  - ▷ chf denotes the main author of this contribution (he can be a novice or an expert).
  - ▷ th denotes the specific topic related to the contribution (we suppose that we have k topics sorted by alphabetical order), where $th \in \{t_1, \ldots, t_k\}$.

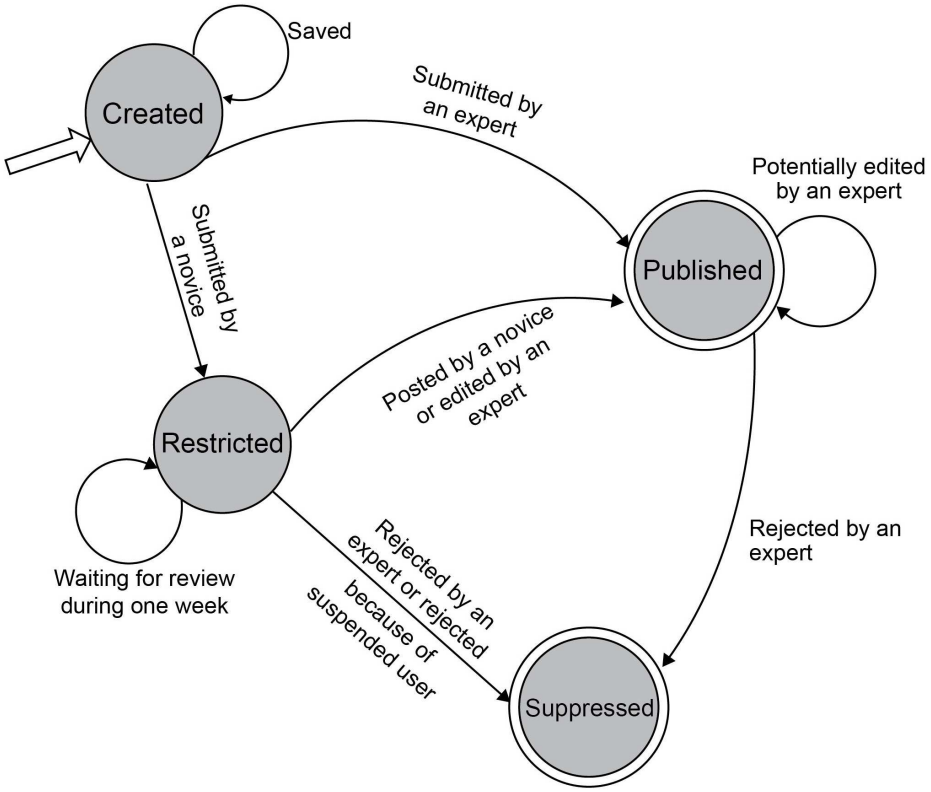


Figure 3.2 The life cycle of a contribution in WiseShare.

- ▷ *vis* denotes the degree of the contribution visibility. Indeed, $vis \in \{created, restricted, published, suppressed\}$. In the case of a contribution created by a novice user, it has a restricted visibility, so it can be only viewed by experts (skilled in the contribution topic). When a contribution is approved by the experts or proposed by one of them, it becomes visible to everyone. When a contribution is suppressed by an expert, it becomes non viewable for all users. The life cycle of a contribution in WiseShare is illustrated in figure 3.2.
- ▷ *Cpl* denotes the set of users identifiers that denounced the content of this contribution.
- ▷ *ctt* denotes the content of the contribution.

- The set of environment attributes $\mathcal{E}$: describes critical information related to the collaborative context in which contributions occur.

  A collaborative environment e is given by the following tuple : $e =< date, Exp, BL >$ where :

  - ▷ *date* denotes the current date of the system.
  - ▷ *Exp* denotes the list of recognized experts sorted by topic.
  - ▷ *BL* denotes the black list which contains only the signature of banned users (ex : the IP addresses of the vandals).

$\mathcal{A}(u)$, $\mathcal{A}(c)$, and $\mathcal{A}(e)$ are attribute assignment relations for user u, contribution c, and environment e respectively :

- ▷ $\mathcal{A}(u) \subseteq idu \times rep \times Skl \times cb \times ncpl$;
- ▷ $\mathcal{A}(c) \subseteq idc \times d_{cr} \times orig \times chf \times th \times vis \times Cpl \times ctt$;
- ▷ $\mathcal{A}(e) \subseteq date \times Exp \times BL$

In our system, an attribute based access control rule (policy) is given as the following predicate :

$\mathcal{P} : \mathcal{U} \times \mathcal{C} \times \mathcal{E} \longrightarrow \{true, false\}$

$$\mathcal{P}(\mathcal{A}(u), \mathcal{A}(c), \mathcal{A}(e)) = \begin{cases} true & \text{if } u \text{ handles } c \text{ in } e \text{ according to } \mathcal{P} \\ false & \text{otherwise} \end{cases}$$

Indeed, given the attribute assignments, the evaluation of an ABAC based policy is reduced to the evaluation of first order logic expressions. By adopting the formalism defined above, we can now express all the requirements of WiseShare according to the ABAC model. The policies controlling the different contributions a user can perform are presented in Table3.1. For instance, the second rule $\mathcal{P}_{publish}(u, c, e)$ states that : a user u can publish a contribution c in a context of collaboration e, if and only if, he doesn't belong to the black list (he is not a vandal), and he is an expert user or a novice user who is the original author of a contribution submitted since more than a week.

3.6 WiseShare Security Enforcement

In this section, we will present the different pseudo-algorithms depicting how a contribution request can affect the different attributes of the system components, especially the contributor attributes. A user in WiseShare can perform one of the following actions : Create, publish, Edit, Suppress and Report. However, before performing such interventions, he must submit a contribution request. If the request is accepted (i.e., satisfies the corresponding policy), he can actually perform his contribution. Otherwise, his contribution will be suspended until the corresponding policy requirements are fulfilled at some moment in the near future. The pseudo-algorithms enforcing WiseShare policies for the different queries are presented in : figure 3.3, figure 3.4, figure 3.5, figure 3.6 and figure 3.7. Based on the system attributes values, each algorithm decides whether the required action will take place or not. In case the required action is executed, the algorithm performs updates on the system attributes accordingly.

Table 3.1 ABAC policy specification in WiseShare

$$\mathcal{P}_{Create}(u,c,e) \Leftrightarrow u.idu \notin e.BL$$

$$\mathcal{P}_{publish}(u,c,e) \Leftrightarrow u.idu \notin e.BL \wedge (u.rep = \text{"}expert\text{"} \vee (u.rep = \text{"}novice\text{"} \wedge e.date - c.d_{cr} \geq 7))$$

$$\mathcal{P}_{Edit}(u,c,e) \Leftrightarrow u.rep = \text{"}expert\text{"} \wedge c.th \in u.Skl$$

$$\mathcal{P}_{Supress}(u,c,e) \Leftrightarrow u.rep = \text{"}expert\text{"} \wedge c.th \in u.Skl$$

$$\mathcal{P}_{Report}(u,c,e) \Leftrightarrow u.idu \notin e.BL \wedge u.idu \notin c.Cpl$$

$$\mathcal{P}_{decide}(u,c,e) \Leftrightarrow u.rep = \text{"}permanent\ expert\text{"} \vee (u.rep = \text{"}expert\text{"} \wedge c.th \in u.Skl)$$

3.7 WiseShare Prototype and Functionalities

In this section, we present the prototype implementing our knowledge sharing collaborative system WiseShare. We have developed this prototype using PHP language with

```
Create(u: user,c:contribution,E: environment,
        name: String, topic: String, h: URL)
{
  c.vis=created;
  c.idc=name;
  c.orig=u;
  c.chf=u;
  c.th=topic;
  c.d_cr=e.date;
  if u.rep= "novice"
    then{
        c.vis="restricted";
        }
    else{
        // c is created by an expert
        c.vis="published";
        }
  // pointer to an html page;
  c.ctt=h;
  Notify all the experts of the list E.Exp
  related to topic to check c;
}
```

Figure 3.3 Effect of creating a contribution

```
publish(u: user,c: contribution, E: environment)
{
 if (c.vis="restricted" AND c.chf== u.idu)
        OR (u.rep= "expert")
    then {
    c.vis="published";
    if u.rep== "novice" then u.cb.(c.th) ++;
    if  u.cb.(c.th) == 500  then
     {
        u.Skl=u.Skl+c.th;
        if u.rep="novice" then
         {
      u.rep="expert";
E. expert=E.expert+u;
           }
     }
}}
```

Figure 3.4 Effect of publishing a contribution

```
Edit(u: user,c: contribution,h: URL)
{
 if major(c.ctt,h)
    then   //c is drastically modified
        c.chf=u;
 c.ctt=h;
 publish(c.chf,c);
}
```

Figure 3.5 Effect of editing a contribution

```
Suppress(c: contribution)
{

 if c.vis= "published" then
   c.chf.cb.(c.th)--;
 c.vis="suppressed";
 if c.chf.rep="expert" AND c.chf.cb.(c.th)==450
    then
     {
      retrieve (c.th) from the list of
      expertise of c.chf.tag;
      If c.chf.Skl is empty
        then
            c.chf.rep="novice";
     }
}
```

Figure 3.6 Effect of suppressing a contribution

```
Report(u: user,c: contribution, e: environment)
{
 c.Cpl=c.Cpl+u.idu;
 c.chf.ncpl++;
 send_complaint(e);

 if (c.chf.rep="novice" AND c.chf.ncpl=20) OR
    (c.chf.rep="expert" AND c.chf.ncpl=100)
    then
    c.chf.rep= "suspended"
}
```

Figure 3.7 Effect of denouncing the content of a contribution

```
decide(u: user,c: contribution, e: environment)
{
 if Relevant_cpl(c) then
   c.chf.rep="vandal"; // deciding that a complaint is credible
 else
 {
  Empty(c.Cpl);
  c.chf.ncpl=0;
  if c.chf.rep= "suspended" the Restore_rep(c.chf);
 }

}
```

Figure 3.8 Effect of handling a complaint about a contribution

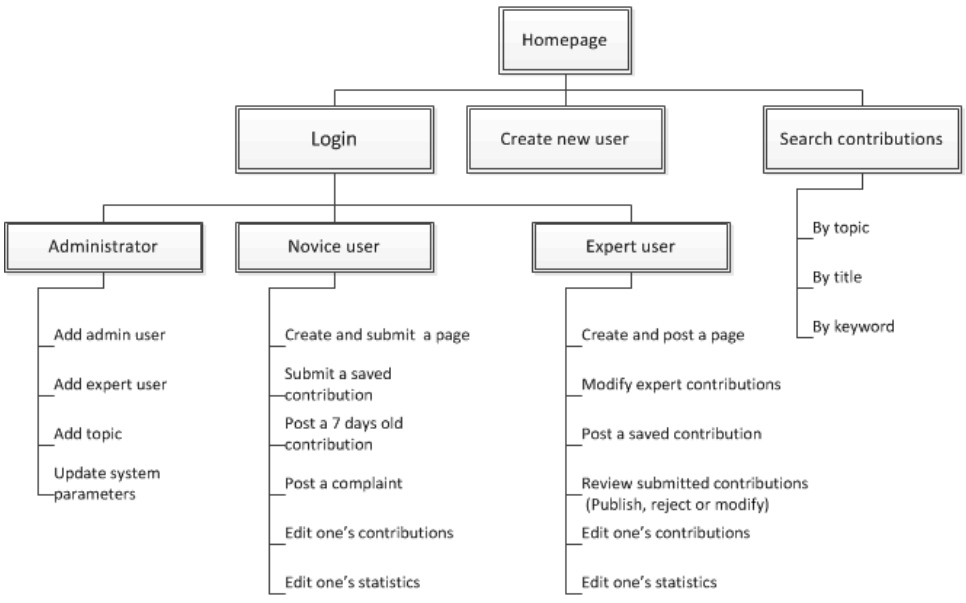


Figure 3.9 The global architecture of WiseShare's prototype

a MYSQL centralized database and an Apache server. Figure 3.9 depicts the global architecture of this prototype. In the following, we will discuss the main functionalities of Wiseshare collaborative system and their implementation in the prototype.

Accessing to the functionalities depends on the profile of the user. As we mention previously in the paper, we have three profiles of users : novice, expert and vandal. For the sake of implementation, a public user profile is needed ; it represents new users exploring the system before registration or users that are not interested by contributing. The functionalities of each user profile are the followings :

- Public users can access the published contributions by searching them by specific criteria (by topic, contribution title or keyword) and, in case they want to contribute, they should first register by creating a user account.
- Each new contributor is tagged as 'novice', novice users have access to the following functionalities :
    - Create a contribution : He can create a contribution in a specific topic (see figure 3.10), save it for further modifications and then submit it for the appreciation of 'expert' users.
    - Publish a contribution : one week after a contribution being submitted, the user can decide to publish this contribution or wait for expert appreciation.
    - publish a complaint against any user having a suspicious behavior.
    - Edit their contribution statistics (see figure 3.11).
- Expert users have access to the following functionalities :
    - Create and publish a contribution. They can also save their contributions for further modifications before publishing them.
    - Review novice users' contributions that are related to their area of expertise. They can publish them with or without modifications and can also reject them.
    - publish a complaint against any user who considered one of his actions as suspicious.
    - Review other experts' contributions that are related to their area of expertise.
    - Edit their contribution statistics.
- Vandal users cannot have access to the system through authentication. Thus, they become public users. However, they can no longer contribute to the system because they belong to the black list.

The implemented prototype of our system has the following advantages :

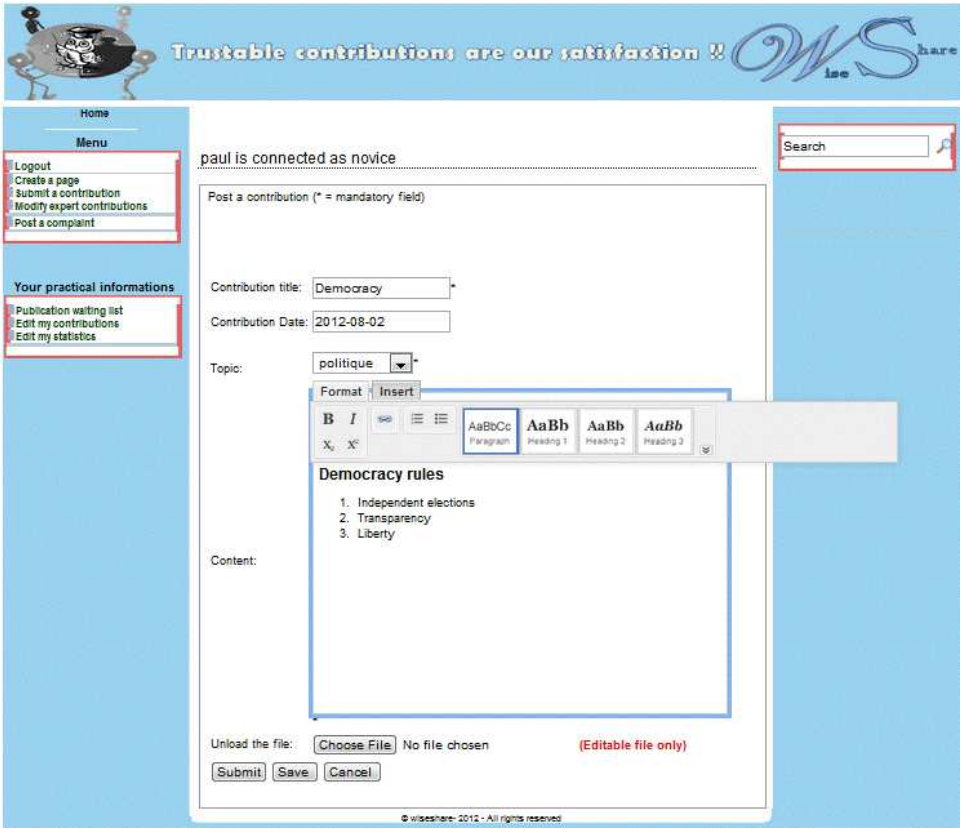


Figure 3.10 Screenshot of contribution creation window.

- Flexible and easy to use : It allows anyone to contribute to the topic he likes provided that he creates an account. Users are endowed with a simple environment for their contribution creation. They can use formatting tools as they wish. The statistics of user's contributions are edited using pie charts.
- Reliable : Unlike other systems such as Wikipedia, user's contributions are verified by experts before being published so that the contents are more reliable. This evaluation could bring contributors to be more serious in the contents they are producing in the system.
- Dynamic : In our system, the more a user is active by creating reliable contributions, the faster he becomes an expert user. Inversely, an expert user can also become a novice if he falls on creating unreliable contributions.
- Interactive : When a novice user submits his contribution, an email is automatically sent to all the experts in the concerned topic. Access to unauthorized functionalities is denied and error messages are displayed to inform the user. That allows those experts to be aware of waiting contributions so that they can evaluate them for publication. Our system also displays a warning when the number of a user's complaints is close to a predefined threshold.
- Automatic : An action taken in our system may automatically change the user's profiles or the object attributes.

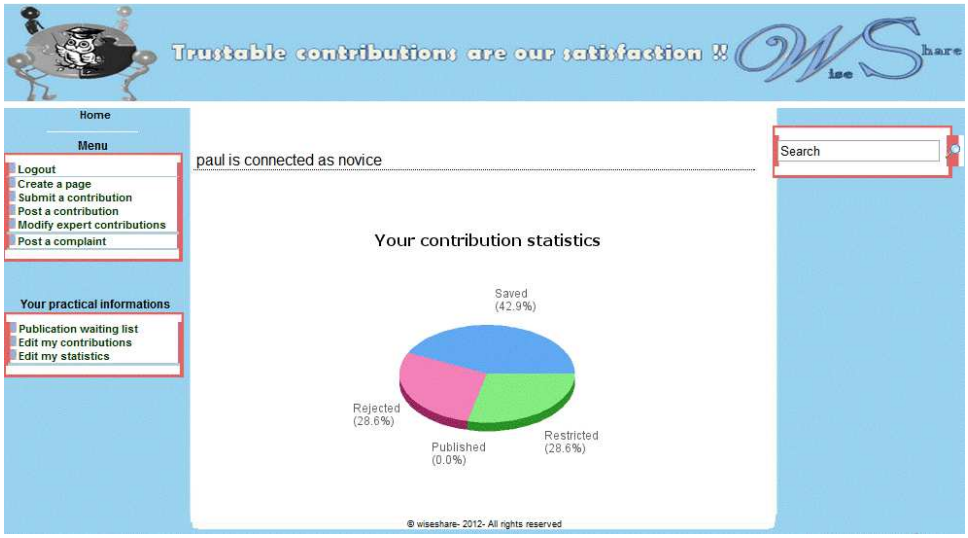


Figure 3.11 Screenshot of contributions statistics window.

3.8 Our model verification

In this section, we describe our model verification design. Before presenting our model design and its abstractions, we present an overview of SMT solvers, particularly $Z3$, the solver we use to verify our ABAC model enforcement. The use of solvers or model checkers in the verification of access control policies is not novel. In (Mondal et al., 2009), the authors proposed a formal technique to perform security analysis on Generalized Temporal RBAC (GTRBAC) model. The model is transformed into timed automata models and liveness and safety properties are then verified against their GTRBAC policy. Unlike their approach, ours proposes to use automata to verify inconsistencies that might occur due to the contributors' actions in the system. In (Schaad et al., 2006), Schaad et al. applied model checking techniques to automated analysis of delegation and revocation functionality. The separation properties are translated into LTL-based constraints and verified using the symbolic model checker NuSMV. But, they did not consider the update problem. In order to tackle this problem, (Hu et al., 2017) presents an approach for assisting administrators to make a desirable update in their RBAC systems. They propose a formalization of the update approach and develop an updating algorithm based on model checking techniques. In (Reith et al., 2009), Reith et al. proposed reduction, optimization and verification techniques that can be used to determine whether security properties are satisfied by Role-based Trust policies. Several works (Mankai and Logrippo, 2005; Jackson, 2012; Schaad and Moffett, 2002; Toahchoodee and Ray, 2005) focus on using Alloy to specify and analyze access control policies.

3.8.1 Z3 SMT overview

According to the authors in (De Moura and Bjørner, 2008), Z3 is an new SMT solver tool from Microsoft Research. This tool aims at checking the satisfiability of logical formulas over one or more theories and is based on first-order logic. A given formula ϕ is said to be *satisfiable* if there exists an interpretation that makes ϕ true. A formula is said to be *valide* if it is true under all structures. For example, $x + y > 3$ is satisfiable under the interpretation $x \mapsto 1$ and $y \mapsto 3$. We report in this paper only some relevant Z3 constructs that we use in our model due to the limit of the paper length. For our model design purpose, we use $Z3Py$, a Python interface for Z3 solver.

- *Sorts.* $Z3$ allows users to declare new data types called sorts (e.g. $a = DeclareSort('a')$).
- *Constants.* They are functions that take no argument (e.g. $b = Const('b', a)$).
- *Functions.* In $Z3$, functions have no side-effect and are defined on all input values (i.e. they are total). The following function $f = Function('f', IntSort(), BoolSort())$ has one integer parameter as input and return a boolean as output.
- *Quantifiers.* Universal quantifier is represented by $ForAll$ and existential one by $Exists$.
- *Solver().* It is used to create a given solver instance (e.g. $s = Solver()$).
- *Add().* This method is used to add constraints to the solver (e.g. $s.add(x <= 8)$)
- *Check().* It checks the satisfiability of all constraints that are associated to the solver (e.g. $s.check()$) and the result returned is either *sat* (satisfiable), or *unsat* (unsatisfiable).

3.8.2 System Design and Specification

Our system design and specifications are based on the requirements of our ABAC access control model. Recall that the goal is to formally verify the robustness of the proposed ABAC policies in the context of contents sharing where many contributors collaborate. In $Z3$ specifications, we proceed as follows :

Environment variables

- *Subjects* : they are declared as array of users represented by their identifier (ID).
- *Objects* : they are declared as array of contributions and are also represented by their ID.
- *Attributes* : they are subjects and objects' attributes and are declared using $EnumSort()$ command.

$$
\begin{aligned}
Right, (create, post, report, edit) = EnumSort('Right','create','post','report','edit'))\\
Topic, (sports, politique, biology, botany, arts) = EnumSort('Topic', ('sports','politique',\\
'biology','botany','arts'))\\
Reputation, (novice, expert, vandal, permanent\_expert) = EnumSort('Reputation',\\
('novice','expert','vandal','permanentexpert'))\\
CtrbStatus, (restricted, published, suppressed) = EnumSort('CtrbStatus', ('restricted',\\
'published','suppressed'))
\end{aligned}
\tag{3.1}
$$

- *links* : from the attributes in 3.1, we declare array of user attributes by linking each occurrence of user with his attributes (i.e. his reputation, connection status, access rights, etc.). We do so with the contributions and each occurrence of contribution has its owner, content topic and status.

$$
\begin{aligned}
uRep &= Array('Urep', User, Reputation)\\
uRight &= Array('uRight', IntSort(), Right)\\
uStatus &= Array('uStatus', IntSort(), BoolSort())\\
ctrbTop &= Array('ctrbTop', IntSort(), Topic)\\
cStatus &= Array('cStatus', IntSort(), CtrbStatus)
\end{aligned}
\tag{3.2}
$$

After variables setting, we declare functions that encapsulate the links between objects or subjects and their attributes.

$$
\begin{aligned}
uCtrb &= Function('uCtrb', IntSort(), IntSort(), BoolSort())\\
uExpTop &= Function('uExpTop', IntSort, Topic, BoolSort())\\
uBList &= Function('uBList', IntSort(), BoolSort())\\
ctrbDelay &= Function('ctrbDelay', IntSort(), IntSort())\\
uNbPubCtrb &= Function('uNbPubCtrb', IntSort(), Topic, IntSort())\\
uNbRejCtrb &= Function('uNbRejCtrb', IntSort(), Topic, IntSort())\\
uNbNegTag &= Function('uNbNegTag', IntSort(), IntSort())\\
uNegTagBy &= Function('uTagBy', IntSort(), IntSort(), BoolSort())
\end{aligned}
\tag{3.3}
$$

The aforementioned functions defined in 3.3 have the following meaning :

— $uCtrb(u, c)$ returns *True* if the user u is the owner of the contribution c and *False* otherwise.

— $uExpTop(u, t)$ returns *True* if the user u is an expert in the topic t and *False* otherwise.

— $uBList(u)$ returns *True* if the user u is in the black list (i.e. he is vandal) and

False otherwise.

— $ctrbDelay(c)$ returns *the number of delay days* of the contribution c waiting for its posting.

— $uNbPubCtrb(u, t)$ returns *the number of published contributions* of the user u in the topic t.

— $uNbRejCtrb(u, t)$ returns *the number of rejected contributions* of the user u in the topic t.

— $uNbNegTag(u)$ returns *the number of negative tags* of the user u.

— $uNegTagBy(u_1, u_2)$ returns *True* if user u_1 receives one negative tag from user u_2 and *False* otherwise.

- Thresholds : in our model, there are several constraints that could impact a user reputation and depending on those constraints, a user reputation can be upgraded or downgraded. Let $nbThresGoodCtrb$, $nbThresBadCtrb$, $nbThresComplNov$ and $nbThresComplExp$ be respectively the threshold of good contributions that allows a novice user to be an expert, the threshold of bad contributions that downgrades an expert to a novice, the threshold of complaints against a novice user and the threshold of complaints against an expert user. The latter two downgrade a user to a vandal.

$$\begin{aligned} nbThresGoodCtrb &= Int('nbThresGoodCtrb') \\ nbThresBadCtrb &= Int('nbThresBadCtrb') \\ nbThresComplNov &= Int('nbThresComplNov') \\ nbThresComplExp &= Int('nbThresComplExp') \end{aligned} \tag{3.4}$$

Constraints

In order to be capable to check the satisfiability of our access control policies, we add some constraints to the solver according to their definition in Table 3.1.

- Solver : let s be the solver defined by $s = Solver()$.

  We apply the constraints mentioned in the access control policies.

- Constraints for $\mathcal{P}_{Create}(u, c, e) \Leftrightarrow u.idu \notin e.BL$

  Let NB be the number of the contributors of the system.

$$\begin{aligned} NB &= Int('NB') \\ u &= Int('u') \\ s.add(And(u >= 0, u < NB)) & \\ s.add(And(Implies(uRight[u] == create, Not(uBList(u))), & \\ Implies(Not(uBList(u)), uRight[u] == create)) & \end{aligned} \tag{3.5}$$

- Constraints for $\mathcal{P}_{publish}(u, c, e) \Leftrightarrow u.idu \notin e.BL \wedge (u.rep = \text{"}expert\text{"}$
  $\vee\ (u.rep = \text{"}novice\text{"} \wedge e.date - c.d_{cr} \geq 7))$

$$
\begin{aligned}
NB &= Int('NB') \\
u &= Int('u') \\
t &= Const('t', Topic) \\
s.add(And(u >= 0, u < NB)) & \\
s.add(And(Implies(uRight[u] == post, Or(And(uRep[u] == expert, Not(uBList(u))), & \\
And(uRep[u] == novice, uCtrb(u,t), ctrbDelay(t) >= 7))), & \\
Implies(Or(And(uRep[u] == expert, Not(uBList(u))), And(uRep[u] == novice, & \\
uCtrb(u,t), ctrbDelay(t) >= 7)), uRight[u] == post))) &
\end{aligned}
\tag{3.6}
$$

- Constraints for $\mathcal{P}_{Edit}(u, c, e) \Leftrightarrow u.rep = \text{"}expert\text{"} \wedge c.th \in u.Skl$

$$
\begin{aligned}
NB &= Int('NB') \\
u &= Int('u') \\
t &= Const('t', Topic) \\
s.add(And(u >= 0, u < NB)) & \\
s.add(And(Implies(uRight[u] == edit, And(uRep[u] == expert, uExpTop(u,t)), & \\
Implies(And(uRep[u] == expert, uExpTop(u,t)), uRight[u] == edit))) &
\end{aligned}
\tag{3.7}
$$

- Constraints for $\mathcal{P}_{Report}(u, c, e) \Leftrightarrow u.idu \notin e.BL \wedge u.idu \notin c.Cpl$

$$
\begin{aligned}
NB &= Int('NB') \\
u_1, u_2 &= Ints('u_1 u_2') \\
s.add(And(u_1 >= 0, u_1 < NB, u_2 >= 0, u_2 < NB)) & \\
s.add(And(Implies(uRight[u_2] == report, And(Not(uBList(u_1)), & \\
uNegTagBy(u_1, u_2, u_1 != u_2), & \\
Implies(And(Not(uBList(u_1)), uNegTagBy(u_1, u_2), & \\
u_1 != u_2), uRight[u_2] == report))) &
\end{aligned}
\tag{3.8}
$$

The aforementioned derived constraints are the main constraints of our access control

policy. However it is possible to check via the $Z3$ solver some different properties of our access control policy. In order to achieve this, we declare some functions that update the attributes of different subjects and objects depending on the actions issued by the users. For example, if a user gets his contribution published that may possibly have an impact on his reputation (i.e. the number of his published contributions in a specific topic becomes greater than the threshold $nbThresGoodCtrb$).

Listing 3.1 'Function for updating user's reputation'

```
def updateUserRep(i,c,t):
    if uCtrb(u,c) and uNbPubCtrb+1 >= nbThresGoodCtrb :
        uRep = Store(uRep, i, "expert")
```

We then add the following constraint into the solver : is there any novice user u who does not become an expert after the number of his published contributions in a specific topic t getting greater than the threshold $nbThresGoodCtrb$?

$$\begin{aligned} NB &= Int('NB') \\ u &= Int('u') \\ t &= Const('t', Topic) \\ s.add(And(u >= 0, u < NB)) & \\ s.add(Exists([u,t], Implies(And(uRep[u] == novice, & \\ uNbPubCtrb(u,t) >= nbThresGoodCtrb), uRep[u] != expert))) & \end{aligned} \tag{3.9}$$

3.8.3 Simulation results and discussion

As we claimed in the section 4) of our paper, the main objective remains a reliable sharing of knowledge where security is strongly enforced. The behavior of each contributor guides his reputation in the system. To reach this goal, we use the SMT solver $Z3$ to declare objects, subjects and their attributes including environment ones. Then, we add all the constraints specified in our access control policies to the solver. Finally, we check the satisfiability of each of them.

For our simulation purposes, we arbitrary set the number of contributors to 100, the number of topics to 5 and at least one expert per topic.

For each constraint, we run the solver using the command $print(s.check())$, we get the following results summarized in Table 3.2.

Table 3.2 Our model simulation results

Constraint No	Simulation results	Comments
(3.5)	*Sat*	A user has the right to *create* a contribution if and only if he is not in the black list
(3.6)	*Sat*	A user has the right to *post* a contribution if he is not in the black list and he is an expert of this contribution topic or the contribution is automatically published after a delay of 7 days.
(3.7)	*Sat*	A user has the right to *edit* a contribution if he is an expert of the contribution topic.
(3.8)	*Sat*	A user has the right to *report* a complaint against another user if the former is not in the black list and he never tags the latter.
(3.9)	*Unsat*	It does not exist in the system a novice user whose reputation does not change to an expert after publishing more than $nbThresGoodCtrb$ contributions.

As we can see, all the rules defined in our access control policy are satisfied and we can conclude that the model is robust.

Since we are using a SMT solver, the complexity of verifying the properties does not increase whenever the number of contributors and their contributions grow because, the rules being verified are defined as first-order logic (FOL) formulas. However, the growth of those rules implies more rules to transform into FOL formulas.

3.9 Conclusion and Future Work

In this paper, we have proposed an attribute-based access based approach for creating and sharing knowledge in collaborative environment. In our system WiseShare, the security model ensuring the reliability of the broadcasted information, is tightly woven. Indeed, each contribution has to obey to a strict access control policy. For this purpose, a contributory profile is built for each user during his activities. This profile reflects as faithfully as possible the expertise and the behavioral history of each contributor. Based on this profile, only enforceable requests are granted. Hence, a user may only perform contributions for which he is skilled. Our ABAC rules guarantee a responsible collaboration and minimizes clumsy contributions and vandalism actions. To achieve this goal, we have defined a formal framework to capture the features of WiseShare components. This formalism allows us to specify our collaborative requirements. In addition, we have expressed the impact of each contribution on updating WiseShare profiles. In order to prove the robustness and the reliability of our ABAC policy, we design a model using the SMT solver $Z3$ and add all the constraints included in our access control policy to the

solver. The solver checking proves that the targeted properties are satisfied. WiseShare may probably lose in performance due to the centralized characteristic of the database. In the future, we plan to integrate time constraints in experts profiles. This will encourage them to propose novel contributions in order to maintain their reputation. We intend also to investigate people-tagging mechanism to take advantage of the opinion of those experts who have effectively demonstrated their integrity throughout their contributions. Finally, we plan to use other model checking techniques and compare them with $Z3$ solver in terms of performance and state-space explosion issue.

CHAPITRE 4
ARTICLE 2 : A FORMAL PROPERTY VERIFICATION FOR ASPECT-ORIENTED PROGRAMS IN SOFTWARE DEVELOPMENT

Moustapha Bande, Hakima Ould-Slimane, Hanifa Boucheneb
Publié à la conférence *20th International Conference on Aspect-Oriented Software Systems, Design, Development and Programming.*

Abstract - Software development for complex systems requires efficient and automatic tools that can be used to verify the satisfiability of some critical properties such as security ones. With the emergence of Aspect-Oriented Programming (AOP), considerable work has been done in order to better modularize the separation of concerns in the software design and implementation. The goal is to prevent the cross-cutting concerns to be scattered across the multiple modules of the program and tangled with other modules. One of the key challenges in the aspect-oriented programs is to be sure that all the pieces put together at the weaving time ensure the satisfiability of the overall system requirements.

Our paper focuses on this problem and proposes a formal property verification approach for a given property from the woven program. The approach is based on the control flow graph (CFG) of the woven program, and the use of a satisfiability modulo theories (SMT) solver to check whether each property (represented par one aspect) is satisfied or not once the weaving is done.

Keywords - Aspect-oriented programming ; Control flow graph ; Satisfiability modulo theories ; Property verification.

4.1 Introduction

Aspect-oriented programming (AOP) (Kiczales et al., 1997) has emerged as a programming paradigm that aims to improve the separation of concerns in software development. In AOP, the system is divided into two parts : the base program containing the main functionality of the system and the aspect program that is composed by the cross-cutting functionality (Elrad et al., 2001). This technique has brought along with it new mechanisms and concepts for implementing crosscutting concerns in a modular manner. Among those crosscutting concerns, security requirements are considered to be more representative since they tend to scatter and tangle with other concerns of a software system. AspectJ (Kiczales et al., 2001) is the most popular AOP language and is based on the Java language. It adds the following new concepts to the Java language :

— **Aspect.** It is a module for handling cross-cutting concerns. It can be seen as a class-like construct.

— **Join point.** It is a well-defined point within a class where a concern is going to be attached during the execution of the program (e.g. method calls, exception thrown,...).

— **Advice.** It is the action taken by an aspect at a particular join point. An advice is implemented as a method of the aspect class. This method is executed *before* or *after* the join point is reached. It may also be executed *around* the join point.

— **Pointcut.** It is a group of join points that need to be matched before running an advice.

— **Weaving.**. It is the process of executing the relevant advice at each join point. This is achieved by a key component called the *aspect weaver*. This component takes the core modules and the aspects and then composes the final program also known as "woven program".

AOP technique helps developers gain in modularity and easy maintenance because this approach isolates crosscutting concerns into aspects. However, in software testing, AOP brings along with it new issues (Alexander et al., 2004) and summarized as follows.

— **Aspects identity and existence.** They do not have independent existence since they usually depend on the context of other classes and also the execution context.

— **Weaving time.** Aspects also tightly linked to the classes they are woven during the execution time. Any change to one of the these classes could have an impact on the aspects.

— **Data and control dependencies between aspects and classes.** During the weaving process, the resulting control flow nor the data flow structure of the woven program is not obvious for the developer .

— **Emergent behavior.** Many faults occur because of the implementation of a given class or aspect, or a side effect of the weaving process of multiple aspects.

— **Changes in normal and exceptional control flow**. Advices containing statements that possibly throw an exception might cause an implicit modification in the system control flow (Ceccato et al., 2005).

With the increasing size and complexity of software systems, most of software development teams need tools or techniques that may help them detect errors or find inconsistencies in the source code. Among the various functionality of a program, security modules are more critical because their violation may lead to enormous loss for enterprises. In AOP, it is not accurate to assume that the successful testing in isolation of the base program and each aspect implies the consistency of the overall system, since interactions and interference usually happen between them and changes in one aspect may have significant impacts on the whole execution of the AOP program.

The problem of detecting errors or inconsistencies during the weaving time has been variously tackled by aspect-oriented community. Some of the previous work have focused on finding approaches and tools for aspect-oriented program testing, using Unified Modeling

Language (UML) models (Massicotte et al., 2005; Badri et al., 2005; Babu and Krishnan, 2009) or graph-based testing (Franchin et al., 2007; Wedyan and Ghosh, 2010; Bernardi, 2008). In this work, we present another approach that combines some of the existing approaches for verifying the satisfiability of properties covered in a woven aspect-oriented program. Every time, one critical aspect related to a specific property is added to the program, it may be possible to check if all the properties being covered by the program remain satisfied. Indeed, such critical requirements need to be handled with accuracy and only an automatic tool that can explore all the necessary states of the program execution could help to avoid the software misbehavior.

The main contribution of this paper is an automatic misbehavior detection of a given property at the weaving time. It is an integrated approach that uses the control flow graph (CFG) of the woven program and then, by transforming the derived CFG into Z3-SMT solver model, we can easily verify the satisfiability of any property. This work can be extended to any new added aspect and also used in collaborative software development in order to automatically verify whether the woven program still satisfies a given property.

The remainder of this paper is organized as follows. In Section 2, we present our formal property verification approach for aspect-oriented programs. Section 3 deals with our model simulation results and Section 4 is dedicated to some previous work in AOP software testing. Section 5 finally concludes the paper and gives some future work.

4.2 Formal property verification in AOP programs

For our design purpose, we use an automatic teller machine (ATM) transaction management system.

4.2.1 Experimental example : an ATM transaction management

As we can see, this example refers to an ATM transaction management in an AspectJ program.

Listing 4.1 Class BankAccount

```
1  package example;
2  import example.BankAccount;
3  import java.io.*;
4  import java.util.Properties;
5
6  public class BankAccount
7  {
8      private String accountOwner;
9      private String accountNumber;
10     private String accountPin;
```

```
11      private double balance;
12
13      public BankAccount(String owner, String account, String pin, double balance) //
            BankAccount constructor
14      {
15        this.accountOwner = owner;
16        this.accountNumber = account;
17        this.accountPin = pin;
18        if (balance > 0.0)
19              this.balance = balance;
20      }
21
22      public void deposit(double depositAmount) // method for deposits
23      {
24        balance = balance + depositAmount; //update the balance
25      }
26
27      public void withdrawal(double withdrawalAmount) // method withdrawal
28      {
29        balance = balance - withdrawalAmount; // update the balance
30      }
31
32      public double getBalance()
33      {
34          return balance;
35      }
36
37      public void setAccount(String account)
38      {
39          this.accountNumber = account;
40      }
41
42      public void setOwner(String owner)
43      {
44          this.accountOwner = owner;
45      }
46      public void setPin(String pin)
47      {
48          this.accountPin = pin;
49      }
50      public String getOwner()
51      {
```

```
52          return accountOwner;
53      }
54
55      public String getAccount()
56      {
57          return accountNumber;
58      }
59      public String getPin()
60      {
61          return accountPin;
62      }
63
64      public boolean authenticate(String userName, String pin) throws Exception{
65
66          File userFile = new File("D:/eclipse/workspace/users.txt");
67          FileInputStream in = new FileInputStream(userFile);
68          Properties properties = new Properties();
69          properties.load(in);
70          String storedPassword = properties.getProperty(userName);
71          in.close();
72          return pin.equals(storedPassword);
73      }
74
75      public static void main(String[] args)
76      {
77        BankAccount account = new BankAccount("Toto","120541263","789",500);
78        account.deposit(500);
79        account.withdrawal(300);
80        }
81  }
```

Listing 4.2 Aspect user's authentication

```
1  package org.example.aop;
2  import example.BankAccount;
3  public aspect AspectAuthentication {
4      pointcut authentication(BankAccount b, double x): (call(*
           BankAccount.withdrawal(..))|| call(* BankAccount.deposit(..))) && target(b)
           && args(x);
5
6      before(BankAccount b, double x) : authentication(b,x) {
7          try {
8              if (!b.authenticate(b.getOwner(),b.getPin()))
```

```
9             {
10            System.out.println("Access denied ... Incorrect pin");
11            System.exit(0);
12            }
13        } catch (Exception e) {
14            System.out.println("Error ... ");
15        }
16    }
17 }
```

Listing 4.3 Aspect transaction checking

```
1  package org.example.aop;
2  import example.BankAccount;
3  public aspect AspectChecking {
4      pointcut checking(BankAccount b, double x): call(* BankAccount.withdrawal(..))
           && target(b) && args(x);
5
6      before(BankAccount b, double x) : checking(b, x) {
7          if (b.getBalance() < x) {
8              System.out.println("Sorry Your account does not have. $" + x + " to
                   withdrawn...");
9              System.exit(0);
10         }
11     }
12
13     after (BankAccount b, double x): checking(b, x) {
14         System.out.println("Transaction successfully done!!! Your current balance is
               : $" + b.getBalance());
15     }
16 }
```

The listing 4.1 shows the Java class *BankAccount* which simulates the functionality of a bank account and represents the *base program*. The aspect defined in Listing 4.2 is used to monitor the access to a bank account by a user. This aspect captures the authentication access control. We define another aspect in Listing 4.3 that captures transaction checking (e.g. being sure that an account has enough money for a withdrawal transaction). Our experimental source code example consists of one class (BankAccount) and two aspects (authentication and transaction checking).

— *Class BankAccount.* This class records user's account information : user name, account ID, pin code and account balance. It also defines methods such as withdrawal and deposit.

— *Aspect AspectAuthentication.* This aspect implements authentication goal. That is giving access to authorized users to their account information. The goal that keeps the information secret from unauthorized people is almost the most common one in information security. In the context of ATM transaction management, violation of such an information security property may be crucial for the bank reputation.

— *Aspect AspectChecking.* It implements a transaction requirement checking. We use a *before* advice to check whether a withdrawal transaction is possible or not considering the target account balance. It also includes a *after* advice that simply displays the target account balance after the transaction has been successfully completed.

Given an aspect-oriented program and any property that may be verified through the execution of this program, we build a fully integrated approach that verifies if the property is satisfied or not during the weaving time.

4.2.2 Our design approach

Our framework in Figure 4.1 uses an existing algorithm that aims to generate the control flow graph of the program. The derived graph is manually transformed into constraints through a SMT solver that is used to verify some security properties.

For our design purpose, we use the algorithm proposed by (Parizi and Ghani, 2008) to generate the CFG of our program. In their work, the authors implemented a tool called *AJcFgraph Builder* that automatically derives an aspect-oriented control flow graph (AOCFG) giving an AOP program. Based on their algorithm, Figure 4.2 shows the CFG of our AOP program and displays the case an access is granted to a user account for doing his transactions. In order to make the graph more readable, we annotate the nodes with the character B followed by the line number of the source code for the *Base* program and A followed by the line number of the source code for the *AspectAuthentication* program and C for *AspectChecking* program . The small b added to the node $B78b$ and $B79b$ stands for *before* advice, whereas small a added to node $B79a$ is used for *after* advice. We also add (in yellow color) nodes and edges that denote the case where the pin code is not valid ($A8 \implies A11 \implies A12$) and the one where account balance is less than the withdrawal amount ($C7 \implies C8 \implies C9$). After the generation of the CFG graph, we build our model by deriving Z3 SMT model from the CFG graph.

4.2.3 Our Z3 model design

Satisfiability Modulo Theories (SMT) is the Satisfiability of formulas with respect to some background theory (Barrett et al., 2009). According to the authors in (De Moura and Bjørner, 2008), Z3 is an new SMT solver tool from Microsoft Research. This tool aims at checking the satisfiability of logical formulas over one or more theories and is based on first-order logic. A given formula ϕ is said to be *satisfiable* if there exists an interpretation

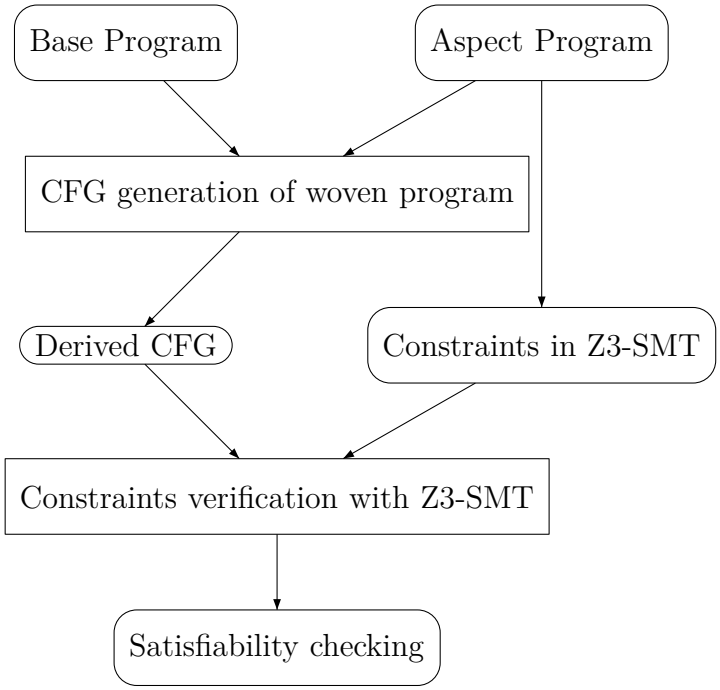


Figure 4.1 Our approach design

that makes ϕ true. A formula is said to be *valide* if it is true under all structures. For example, $x + y > 3$ is satisfiable under the interpretation $x \mapsto 1$ and $y \mapsto 3$. We report in this paper only some relevant Z3 constructs that we use in our model due to the limit of the paper length. For our model design purpose, we use $Z3Py$, a Python interface for Z3 solver.

— *Sorts.* Z3 allows users to declare new data types called sorts (e.g. $a = DeclareSort('a')$).

— *Constants.* They are functions that take no argument (e.g. $b = Const('b', a)$).

— *Quantifiers.* Universal quantifier is represented by $ForAll$ and existential one by $Exists$.

— *Solver().* We can use it to create a given solver instance (e.g. $s = Solver()$).

— *Add().* We can use this method to manually add constraints to the solver (e.g. $s.add(x <= 8)$)

— *Check().* We can call this method to check the satisfiability of all constraints that are associated to the solver (e.g. $s.check()$) and the result returned is either *sat* (satisfiable), or *unsat* (unsatisfiable).

In order to derive the Z3 model from the CFG of the woven program, we use the rules summarized in Table 4.1.

— We declare in $Z3Py$ a graph that is a representation of the CFG derived from the woven program. We then add the following attributes to nodes and edges to capture data information during the simulation of the model :

  — *Node* : this attribute captures the name of a node in the graph.

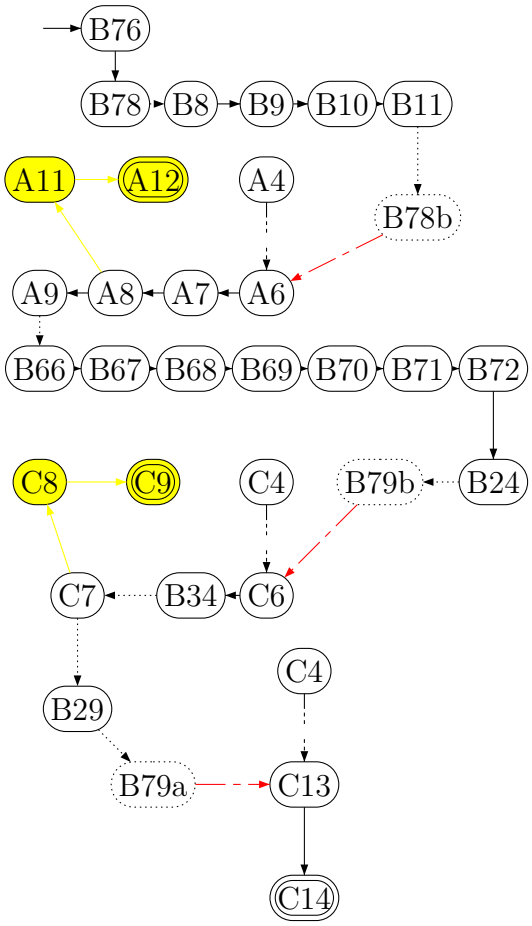
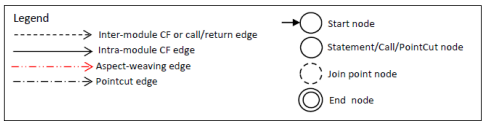


Figure 4.2 The derived CFG of our program



- *ExecStatus* : this attribute captures the execution status of an edge in the graph.
- *isJoinPoint* : this attribute return true if the node is a join point node and false otherwise.
- *isAspectWeavingEdge* : this attribute return true if the edge is an aspect-weaving edge and false otherwise.
- *isPointcutEdge* : this attribute return true if the edge is an pointcut edge and false otherwise.

— We define the function $isAuth(u, p)$ that returns $true$ if the user u enters a valid pin p and $false$ otherwise, giving the attributes of the edge and nodes being traversed in the graph. This corresponds to the advice of the aspect program. It is declared using $isAuth = Function('isAuth', IntSort(), IntSort(), BoolSort())$.

— For our simulation purpose, we define an array variable that contains 3-tuple of

user, his pin code and his account balance. In this array variable, are stored random generated triples of user, pin and balance.

Table 4.1 From CFG to Z3 - Transformation rules

	CFG	Z3
1	Set of nodes Nd	Nd = DeclareSort('Nd')
2	A node $nd_1 \in Nd$	nd_1 = Const('nd_1',Nd)
3	Set of edges Eg	Eg = DeclareSort('Eg')
4	An edge $eg(nd_i, nd_j) \in Nd \times Nd$	eg = Function('eg',Nd,Nd,BoolSort())
5	$isPointCut(nd_i)$	isPointCut = Function('isPointCut',Nd,BoolSort())
6	$isWeavingEdge(nd_i, nd_j)$	isWeavingEdge = Function('isWeavingEdge',Nd,Nd,BoolSort())
7	$isJointPointEdge(nd_i, nd_j)$	isJointPointEdge = Function('isJointPointEdge',Nd,Nd,BoolSort())

4.3 Properties verification

We recall that the goal of our model is to automatically verify properties for an AOP program. Based on the parameters used to run the main method, we verify whether an unauthorized user could access or not the ATM system. After building our Z3 model, we add some constraints into it and then check its satisfiability.
First of all, we add the constraint

$$s.add(ForAll([u_i, p_i], Implies(Not(isAuth(u_i, p_i)), g.es[11]["ExecStatus"]))). \tag{4.1}$$

The checking of the solver using $print(s.check())$ outputs the result Sat meaning that our model is *satisfiable*. This shows that for any given user u_i and his pin code p_i, if the pin code is not valid then we do have an edge between nodes $A11$ and $A12$. The attribute *"ExecStatus"* of this edge is then *True* corresponding to the edge whose index is 11. Consequently, the access is denied and the end node is $A12$. We output only the part of the graph with edges having their execution status at *true* in Figure 4.3. We also add to the solver this constraint

$$s.add(ForAll([u_i, p_i, b_i, x], Implies(And(isAuth(pi, fi), x <= b_i), g.es[32]["ExecStatus"]))). \tag{4.2}$$

Then, we run the model using $check()$ method and the output result of the model checking is Sat, that is the user u_i has entered a valid code pin and the withdrawal amount is less than his account balance, so that the transaction is successfully completed. Thus, the

```
====== RESTART: D:/z3-4.6.0-x64-win
/bin/python/Cfg_verification_ATM.py
======
sat
B76 --> B78
B78 --> B8
B8 --> B9
B9 --> B10
B10 --> B11
B11 --> B78b
B78b --> A6
A4 --> A6
A6 --> A7
A7 --> A8
A8 --> A11
A11 --> A12
>>>
```

Figure 4.3 The model checking output with constraint (4.1)

attribute *"ExecStatus"* of the edge whose index is 32 is *True*. Consequently, the end node in the output in Figure 4.4 is $C14$. Finally, We add to the solver the following constraint that verifies whether the user does or not earn enough money to make his withdrawal transaction.

$$\begin{aligned} s.add(ForAll([u_i, p_i, b_i, x], Implies(And(isAuth(pi, fi), \\ x > b_i), g.es[27]["ExecStatus"]))). \end{aligned} \tag{4.3}$$

Then, we run the model using *check*() method and the output result of the model checking is *Sat*, that is, the user has access to the ATM system but his account balance is not enough to make his withdrawal transaction. Thus, the attribute *"ExecStatus"* of the edge whose index is 27 is *True*. Consequently, the end node in the output in Figure 4.5 is $C9$. We note that the increase of the number of users does not affect the size of our model since the system follows the execution of the *main*() method. In addition, it is also possible to define other types of properties and add them to the model and let Z3-SMT solver checks the satisfiability of the whole system. In Z3, it is possible to handle function composition, that is a function call in the CFG that also calls other functions. We could also add this kind of constraint to the model and check the solver satisfiability. That helps to make modular verification and also incremental one. Our model is flexible and could be used whenever a developer adds a new aspect that captures a specific concern. It can be extended to a collaborative environment where developers together add components in a base program.

4.4 Related work

Many researches have focused on software testing using several formal methods and tools to detect faults and errors during compilation or running time. These approaches can be classified into three categories :

```
====== RESTART: D:/z3-4.6.0-x64-win/bin
/python/Cfg_verification_ATM.py ======
sat
B76 --> B78
B78 --> B8
B8 --> B9
B9 --> B10
B10 --> B11
B11 --> B78b
B78b --> A6
A4 --> A6
A6 --> A7
A7 --> A8
A8 --> A11
A11 --> A12
A8 --> A9
A9 --> B66
B66 --> B67
B67 --> B68
B68 --> B69
B69 --> B70
B70 --> B71
B71 --> B72
B72 --> B24
B24 --> B79b
B79b --> C6
C4 --> C6
C6 --> B34
B34 --> C7
C7 --> C8
C8 --> C9
C7 --> B29
B29 --> B79a
B79a --> C13
C4 --> C13
C13 --> C14
>>>
```

Figure 4.4 The model checking output with constraint (4.2)

The first category is related to those approaches that use model checking to verify whether a given property is verified or not. Among them, in (Fu et al., 2009), the authors proposed an aspect-oriented Petri net with Aspect-Oriented Software Development (AOSD) mechanisms. They developed an automated approach for formally analyzing the software design using the model checking technique and analysis tool PROD. In (Xu et al., 2012), Xu et al. presented a framework called Model-based Aspect/class Checking and Testing (MACT) that aims to test the conformance of aspect-oriented programs against their aspect-oriented state model. The framework also uses model checking for test generation from counterexamples. Fradet et al. (Fradet and Hong Tuan Ha, 2010) proposed a formal framework to enforce availability properties on services sharing resources. The authors used timed automata in order to express and enforce properties on execution time. In (Ubayashi and Tamai, 2002), they defined an automatic verification approach using model checking that verifies the correctness of AOP-based programs. In order to check the

```
====== RESTART: D:/z3-4.6.0-x64-win
/bin/python/Cfg_verification_ATM.py
======
sat
B76 --> B78
B78 --> B8
B8 --> B9
B9 --> B10
B10 --> B11
B11 --> B78b
B78b --> A6
A4 --> A6
A6 --> A7
A7 --> A8
A8 --> A11
A11 --> A12
A8 --> A9
A9 --> B66
B66 --> B67
B67 --> B68
B68 --> B69
B69 --> B70
B70 --> B71
B71 --> B72
B72 --> B24
B24 --> B79b
B79b --> C6
C4 --> C6
C6 --> B34
B34 --> C7
C7 --> C8
C8 --> C9
>>>
```

Figure 4.5 The model checking output with constraint (4.3)

correctness of the woven program, they defined a property in Computation Tree Logic (CTL) and represent this property by an aspect that is finally used for the property verification. In (Denaro and Monga, 2001), Denaro et al. proposed an approach based on analyzing aspect-based software components in order to verify safety properties. The aspects are modeled using a PROMELA process template and the derived model is analyzed with the model checker SPIN in order to verify the deadlock problem of the synchronization policy. Another state-based approach has been proposed in (Xu et al., 2007) for modeling the specification of an aspect-oriented design. In this paper, the authors used a Labeled Transition System Analyzer (LTSA) model checker to verify the generated finite state processes against the desired properties. In (Stolz and Bodden, 2006), a runtime verification framework for Java programs has been proposed. They instrumented Java bytecode with Linear-time Temporal Logic (LTL) formula and for the properties verification purpose, they used an automaton-based approach where transitions are implemented and triggered using aspects. They also developed a prototype called Java Logical Observer (JLO) that gives a way to derive a verification aspect from the formula. The technique

used in this paper, however, implies injecting aspects in order to verify a specific property. That can become more challenging while the number of properties being verified grows. These papers did not, however, describe a modular verification methodology or most of them use source code instrumentation for the verification purposes.

The second category of related work is data-flow or control-flow based testing approaches. In (Lemos and Masiero, 2011), a pointcut-based coverage analysis approach using structural model based on Java bytecode has been proposed. In their approach, the authors used a control and data flow graph for their testing criteria. They also proposed a series of control flow and data flow testing criteria based on a PointCut-based Def-Use (PCDU) graph. However, they did not mention the data flow between integrated units among their testing criteria. In (Zhao, 2003), the author proposed a data-flow-based unit testing approach in which three levels of testing are performed : intra-module, inter-module and inter-aspect or inter-class testing. The model is based on modeling a control graph flow for only the program classes and is extended to the woven program. Finally, the program issues are analyzed by computing def-use pairs of a class or an aspect. This work, however, does not focus on the advice interactions. Fradet et al. (Fradet and Hong Tuan Ha, 2010) proposed a formal framework to enforce availability properties on services sharing resources. The authors used timed automata in order to express and enforce properties on execution time for preventing denial of service attacks. However, they did not consider any change that may occur in the source code. In (Lemos et al., 2007), a derivation of control flow and data flow based testing criteria for aspect-oriented programs has been defined. This model is used to support structural testing to unit testing of AspectJ programs. The key problem with existing dataflow test criteria is the difficulty in covering all types of data flow interactions for AOP programs. Harlem et al. (Hamlen and Jones, 2008) proposed an aspect-oriented declarative security policy specification language for in-lined reference monitoring. This language called SPOX (Security Policy XML) is an XML-based security policy specification in which policies denote an aspect-oriented security automaton.

In the first category of the aforementioned work, the authors instrumented the source code by injecting pieces of code (e.g. aspects) in order to verify a specific property. In the second category, testing criteria are defined on control flow without taking into account the data flow or the core program and the aspects are tested separately. Unlike the above work, our paper focuses on verifying a security property (e.g. authentication, privacy, etc.) encapsulated in an aspect for a woven program and also captures the interactions between the base program and the aspects.

The third category is UML-based transformation using sequence diagrams. Hossain et al. (Hossain et al., 2017) proposed a transformation technique called "Zigzag transformation" to introduce aspects at the architecture level in order to verify properties of the old architecture versus the ones introduced by the aspects. Bowles et al. (Bowles et al., 2016) introduced a novel formal automated technique for weaving aspects using Z3-SMT solver.

Their technique is based on UML sequence diagrams to capture the base and the advice and pointcut models are transformed into equivalent representations using Labelled Event Structures (LES). They also compared the performance using Z3 with their earlier work that used Alloy (Bowles et al., 2016) for sequence diagram composition. Tahara et al.(Tahara et al., 2017) proposed a tool called "CAMPer" that uses Maude specification language to express dynamic aspect weaving. This tool takes an input UML class diagram, sequence diagrams and a Linear Temporal Logic (LTL) formula and automatically verifies the models.

Unlike the aforementioned work, our approach combines control flow graph generation and a SMT solver to check the satisfiability of properties by deriving a Z3 SMT model from the CFG.

4.5 Conclusion and future work

In this paper, a formal verification of properties in aspect-oriented programming is proposed. The main idea of our approach is to generate a control flow graph from a giving woven aspect-oriented program and to derive Z3-SMT model in order to verify some critical properties such as confidentiality. Our approach is flexible and can be extended to both AOP programs with many aspects that encapsulate security property or specific requirement and collaborative software development environments where many developers can collaborate in the same project. In such an environment, this formal verification technique could be used to verify critical properties whenever any developer adds or modifies a piece of code (e.g. aspects, base program).

As future work, we are planning to extend this work to a collaborative software development including dealing with many aspects. Finally, we will look into building a tool that can automatically generate Z3-SMT model from the CFG.

CHAPITRE 5
ARTICLE 3 : CONSISTENT OPERATIONAL TRANSFORMATION APPLIED TO COLLABORATIVE CODING EDITORS

Moustapha Bande, Hanifa Boucheneb
Soumis pour publication au journal *Journal of Systems and Software.*

Abstract - Complex software development involves nowadays about multiple programmers located in different areas working together on the same development project in order to achieve efficiency, improve productivity and reduce development time. Replicated architecture is one of the architectures that is used to offer a development environment to such a high-level collaboration work. However, one of the key challenges of such an environment is to achieve consistency maintenance of all the replicas due to the automatically propagation of any local change of the shared source code. In this paper, we propose a new operational transformation (OT) approach by defining a new inclusive transformation (IT) function that can be used in software coding editor. Then, we prove that this IT function ensures consistency maintenance by using a model checking technique and Difference Bound Matrices.

Keywords - Real-time collaborative coding ; operational transformation ; difference bound matrices ; model checking ; convergence property ; proof of convergence

5.1 Introduction

Software development for large and complex system becomes nowadays a highly collaborative process wherein working together to reach a goal is very efficient and cost-effective. The productivity of developers and the quality of their achieved products highly depend on the efficiency and the effectiveness of their collaboration during the programming work. Through the collaboration process, software developers work together on the same source code based on two main strategies : editing the same copy of source code based on an integrated development environment (IDE) or editing distinct copies in parallel. In this paper we focus on the latter where each developer has his own copy of source code and the updates are propagated from one site to others in real-time without the need of a central server. Two main techniques are usually used in order to achieve collaboration : non-real-time programming collaboration and real-time programming collaboration.

In the former category of collaboration, multiple programmers edit the same objects (e.g., files and directories of source code) and manually merge changes by using a version control system such as subversion (Collins-Sussman, 2002) and concurrent version system (CVS)

(Berliner et al., 1990). In such a kind of collaboration, each programmer works independently locally before merging his updates that can now be visible by other members of the collaboration.

In the latter category, programmers participating in the collaboration edit the same artifacts concurrently and instantly propagate the updates to others. In order to achieve higher responsiveness and unlimited interaction between users, replicated architecture is widely used where local operations are immediately executed and remote ones need to be transformed before being executed. Since the operations may be issued concurrently without ordering, that may lead to divergence when the updates are applied to each site. Many techniques have been used to ensure consistency between all the copies after the execution of all the operations issued locally. Among them, we have Multi-version (MV) (Bernstein and Goodman, 1983), Serialization-Resolution of Conflicts (SRC) (Ellis and Gibbs, 1989), Commutative Replication Data Type (CRDT) (Preguica et al., 2009) and Operational Transformation (OT) (Sun and Ellis, 1998; Sun et al., 1998) approach.

The OT approach is the one we focus on in this paper approach and its basic idea is the transformation of an operation before its execution against other previously executed independent and concurrent operations. When dealing with OT, we face three major inconsistency issues :

— **Divergence.** The difficulty relies on maintaining the consistency of all replicas.

— **Causality violation.** Refers to the non-respect of the performed operations order (i.e. violation of the cause-effect principles).

— **Intention violation.** The intention of an operation does not remain valid after its execution.

Several researches have been conducted to find algorithms or inclusive transformation (IT) functions that may lead to the consistency maintenance. To the best of our knowledge, only few researches have focus on finding a consistent IT function in a real-time collaborative coding editor system. Most of them are related to text editing. Therefore, we propose a novel IT function that ensure convergence and can be applied to both text editing and code editing in a peer-to-peer environment.

The remainder of the paper is organized as follows. Section 2 relies on an overview of OT approach. Section 3 presents some previous work in real-time collaborative software editing systems. While Section 4 is dedicated to our IT function based on absolute positions. Section 5 deals with the proof of correctness of our IT function. In Section 6, we provide some limitations of our approach. Section 7 finally concludes this paper and provides an outlook of future work.

5.2 Operational Transformation Overview

5.2.1 Operational Transformation approach

In operational transformation approach, we have n sites and each site owns a copy of the shared source code which is a finite sequence of source code lines. We assume that the shared source code can be modified by each site using only the following primitive operations :

$$\mathcal{O} = \{Ins(p, l) | l \in \mathcal{L} \text{ and } p \in \mathbb{N}\} \cup \{Del(p) | p \in \mathbb{N}\} \cup \{Nop\}$$

where $Ins(p, l)$ inserts the line of source code l at position p; $Del(p)$ deletes the line of code at position p, and Nop is the idle operation that has null effect on the shared source code. Since each site has his local copy of the shared program, a local state ls is altered only by operations executed locally.

The main problem in operational transformation approach is the *consistency maintenance* (or *convergence*) of all replicas. Let us take an example to illustrate more this divergence problem. Consider a team of two programmers (two sites) working on a shared source code represented by a sequence of source code lines. Both hold locally the basic source code shown in Fig.5.1. Initially, both copies hold the basic source code *"a=0; b=a+1; c=a+b"*. Site 1 executes operation $o_1 = Ins(1,$ *"a=a+1"*$)$ to insert the line *"a=a+1"* at position 1 (*i.e.* line 1). Concurrently, site 2 performs $o_2 = Del(2)$ to delete the instruction line *"c=a+b"* at position 2. When o_1 is received and executed on site 2, it produces the expected source code *"a=0; a=a+1;b=a+1"*. But, when o_2 is received on site 1, it does not take into account that o_1 has been executed before it and it produces the string *"a=0;a=a+1;c=a+b"*. The result at site 1 is different from the result of site 2 and it apparently violates the intention of o_2 since the third line of source code *"c=a+b"*, which was intended to be deleted, is still present in the final source code. Consequently, we obtain a *divergence* between sites 1 and 2. It should be pointed out that even if a serialization protocol (Ellis and Gibbs, 1989) was used to require that all sites execute o_1 and o_2 in the same order (*i.e.* a global order on concurrent operations) to obtain an identical result *"a=0; a=a+1;c=a+b"*, this identical result is still inconsistent with the original intention of o_2.

In order to maintain convergence, the *Operational Transformation* (OT) approach has been proposed by (Ellis and Gibbs, 1989). When a site i gets an operation o that was previously executed by a site j on his replica of the shared object, the site i does not necessarily integrate o by executing it "as is" on his replica. It will rather execute a variant of o, denoted by o' (called a *transformation* of o) that *intuitively intends to achieve the same effect as o*. This transformation is based on an *Inclusive Transformation* (IT) function.

As an example, Fig.5.2 illustrates the effect of an IT function on the previous example. When o_2 is received on site 1, o_2 needs to be transformed according to o_1 as follows :

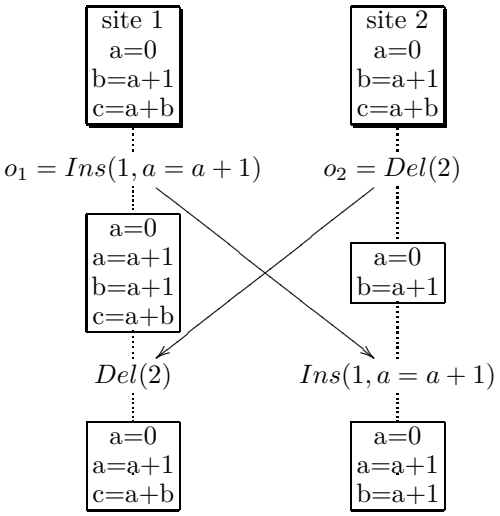


Figure 5.1 Incorrect integration.

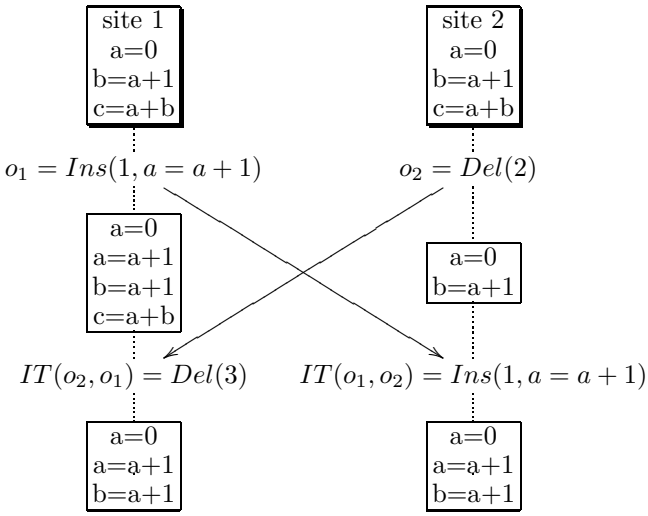


Figure 5.2 Integration with transformation.

$IT(Del(2), Ins(1, a{=}a{+}1)) = Del(3)$. The deletion position of o_2 is incremented because o_1 has inserted a line at position 1, which is before the line deleted by o_2. Next, o_2' is executed on site 1. In the same way, when o_1 is received on site 2, it is transformed as follows : $IT(Ins(1, a{=}a{+}1), Del(2)) = Ins(1, a{=}a{+}1)$; o_1 remains the same because *"a=a+1"* is inserted before the deletion position of o_2.

In the literature, many researchers have proposed several IT functions : *Ellis*'s algorithm (Ellis and Gibbs, 1989), *Ressel*'s algorithm (Ressel et al., 1996), *Sun*'s algorithm (Sun et al., 1998), *Suleiman*'s algorithm (Suleiman et al., 1997) and *Imine*'s algorithm (Imine et al., 2003). Some of them have been widely used in collaborative editors to ensure consistency maintenance. Before proposing our IT function, let us define consistency criteria.

5.2.2 Consistency criteria

An OT-based collaborative editor is said to be *consistent* iff it satisfies the following properties :

1. *Causality preservation :* for any pair of updates u_1 and u_2, if $o_1 \rightarrow o_2$ then o_1 is executed before o_2 at all sites.
2. *Convergence :* when all sites have performed the same set of updates, the copies of the shared document are identical.

To preserve the causal dependency between updates, timestamp vectors are used. In (Ressel et al., 1996), the authors have established two properties $TP1$ and $TP2$ that are necessary and sufficient to ensure data convergence for *any number* of operations executed in *arbitrary order* on copies of the same object : For all o_1, o_2 and o_3 pairwise concurrent operations :

- $TP1 : [o_1 ; IT(o_2, o_1)] \equiv [o_2 ; IT(o_1, o_2)]$.
- $TP2 : IT^*(o_3, [o_1 ; IT(o_2, o_1)]) = IT^*(o_3, [o_2 ; IT(o_1, o_2)])$.

Property $TP1$ defines a *state identity* and ensures that if o_1 and o_2 are concurrent, the effect of executing o_1 before o_2 is the same as executing o_2 before o_1. Property $TP2$ ensures that transforming o_3 along equivalent and different operation sequences will give the same operation. Accordingly, by these properties, it is not necessary to enforce a global total order between concurrent operations because data divergence can always be repaired by operational transformation. As we mentioned before, in (Randolph et al., 2015), the authors proved that Sun's IT (the one used by most of the above work) does not ensure convergence in a distributed environment. Let us apply their IT function to the following example :

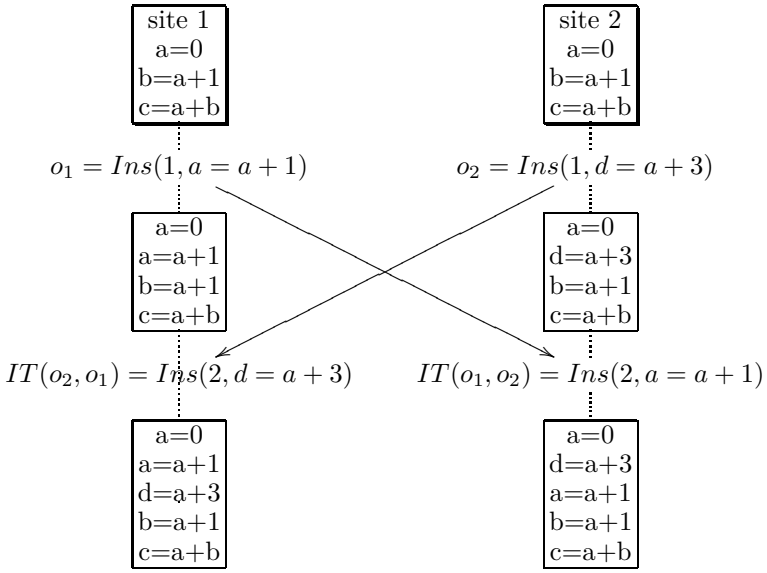


Figure 5.3 Violation of TP1 for Sun'IT.

As we can note in the Fig. 5.3, Sun's IT function violates the property TP_1 and leads to divergence. For example, the value of the variable d in site 1 is 4 but in site 2, its value is 3, leading to data inconsistency. Unlike Sun's IT function, in (Randolph et al., 2015), the authors found an IT that satisfies both $TP1$ and $TP2$. However, finding an IT function that satisfies $TP1$ and $TP2$ is considered as a hard task in peer-to-peer software code editing, because this proof is often unmanageably complicated.

5.3 Related work

Many works have been done on building environments and systems that facilitate real-time collaboration on software development. These systems are mainly based on two environments : client-server and distributed environments.

5.3.1 Client-server environment

In this category of work, we have *Collabode* (Goldman et al., 2011), a web-based IDE that supports collaborative development through real-time code sharing in a client-server

environment. The server hosts and manages the projects by using Eclipse. It holds two versions of the source code : 1) a *union version* seen and edited by all the users ; 2) a disk version that is updated during the compilation time and that contains all the users' edits. After the compilation, only users' methods that no longer contain error are displayed in the disk version as well as in the union version. Another client-server solution is *CoRED* (Lautamäki et al., 2012), a web-based collaborative real-time editor for Java applications. In order to handle concurrent edits, *CoRED* uses Differential Synchronization with shadows (Fraser, 2009) consisting in maintaining a shadow copy of the shared document along with the editing copy on both the server and the client. Myers's algorithm (Myers, 1986) is used to calculate the difference between any changes in the editing copy and the latest copy of the shadow copy. These differences are sent to the server as a patch that is applied to the shared document and keeps the server shadow copy in sync. *CoRED* offers also locking technique that can be used to lock a portion of the shared document for an exclusive editing. Applying multiple patches to the shared document is a heavy operation and locking a portion of the source code may lead to restricting the collaboration. *Idaho Collaborative IDE* (Integrated Dvelopment Environment) named *ICI* (Bani-Salameh et al., 2008) also uses client-server architecture and allows developers to collaborate in real-time. *ICI* has four main components : 1) a collaborative editor, 2) a collaborative shell, 3) a set of communication tools such as a text and voice chat, and 4) an interface for collaboration control, which allows one user to invite other developers to enter the collaborative session. The first component allow developers to share source code editing. *ICI*'s real-time collaborative editor is based on two modes : watch and edit. Only one user can use the edit mode one at the same time, the others being in the watch mode, meaning that it does not support concurrent changes on the shared program.

Another collaborative environment is *Any-Time Collaborative Programming Environment* (ATCoPE) (Fan et al., 2012) that seamlessly integrates real-time and non-real-time collaboration tools. It also uses a client-server architecture and developers taking part of the collaboration first log into the server via two sessions : non-real-time session and real-time session. In the former, source code is downloaded from the server to the *ATCoPE* client inside the developer private workspace. Any change to the local copy is invisible to the public until a commit is manually performed by the local developer. Unlike the non-real-time session, changes made by a real-time session participants in their local copy are automatically visible to all the developers participating in the same session. When conflicts occur during concurrent editing, they are solved using techniques such as *operational transformation* (OT) using *Sun's* Inclusive Transformation (IT) function (Sun and Ellis, 1998) ; and *Dependency-based Automatic Locking* (DAL) (Fan and Sun, 2012b).

CodeR (Kurniawan et al., 2015) is another real-time code editor application for collaborative programming. It is a web application that uses Facebook plug-in to allow users to work on the same programming project while communicating in real-time via chat media.

5.3.2 Distributed environments

In this distributed environment area, we have *D-Praxis* (Mougenot et al., 2009), a peer-to-peer collaborative model editing framework based on optimistic replication. Each developer participating in the collaboration has his own copy and changes made on the local copy are automatically propagated to others. *D-Praxis* proposes a protocol for changes propagation and a mechanism to detect and solve conflicts due to concurrent edits. To deal with conflicts, *D-Praxis* uses two principles : 1) *Lamport's clock* (Lamport, 1978) that extends the "happened before" relation between issued operations to a total consistent ordering of those operations and 2) when two operations are in conflict, the latter is kept and the former is canceled unless it is a delete operation. Even though, this approach is simple and efficient according to the authors, we must point out that it does not preserve developers' intention and is more restrictive. To extend *D-Praxis*, *C-Praxis* (Michaux et al., 2011) has been proposed. It is a collaborative modeling framework for developers. It is based on *Telex* (Benmouffok et al., 2009) a middleware layer for collaborative application software. Telex handles partial replication and conflict detection. In order to deal with the consistency maintenance issue that may arise from concurrent edits, Telex runs a consensus protocol between sites and chooses the common sound prefix proposed by each site. If a consensus cannot be found, Telex makes an arbitrary choice. This framework is used for asynchronous changes and it does not preserve user intention and it may happen that some of the concurrent operations are ignored. *CoEclipse* (Fan and Sun, 2012a) is another real-time collaborative programming system that converts a single-user Eclipse IDE into multi-user real-time programming tool. A CoEclipse client includes an Operational Tranformation module, a Dependency-based Automatic Locking (DAL) module, an awareness module and a communication module. *CoEclipse* supports syntactic and semantic consistency maintenance. The former is achieved by using operational transformation approach and utilizing Inclusive Transformation (IT) function, the one proposed in (Sun et al., 1998). The latter is achieved using a DAL technique (Fan and Sun, 2012c). That is, two users are not allowed to edit the same segment of source code and the dependent segment at the same time (*mutual exclusion*). Locks are granted automatically by the DAL module.

In their work, (Randolph et al., 2015) showed that all the inclusive transformation (IT) functions prior to their work including *Sun'IT* function (the one used by most of the above work), do not ensure data convergence in a peer-to-peer environment and they found an counterexample for each of them. Considering this, they proposed another IT function that adds a new parameter to the signature of the *insert* operation. This parameter is the number ND of symbols deleted before the inserting position. This parameter is incremented whenever the insert operation is transformed against a delete operation and the deleting position is smaller than the inserting one. They proved the consistency of the proposed IT function using symbolic model-checking technique.

However, according to their *Lemma*1, *any pair insert operations generated on the same state, the order relation of their positions is the same as the order of their extra parameter nd*, that is $p_1 < p_2 \implies np_1 \leq np_2$. As we can see, that may lead to a hard work when trying to undo all the transformations made during the OT process since the reverse $np_1 < np_2 \implies (p_1 < p_2) \vee (p_1 = p_2)$ implies a non deterministic choice. In (Nicolaescu et al., 2016), the authors propose *Yet Another Transformation Approach* (YATA) for peer-to-peer shared editing web applications that ensures convergence and intention preservation. They define an algorithm that uses a linked list to represent linear data. An insert operation is defined by $o_k(id_k, origin_k, left_k, right_k, isDeleted_k, content_k)$ where id_k is the insert operation unique identifier, $left_k$ and $right_k$ refer to already existing insertions (i.e., the previous node and the next node in the list). $content_k$ is the content being inserted, $isDelelted_k$ marks an insertion as deleted. $origin_k$ is the direct predecessor at creation time and never changes when new insertions are executed. For conflicting insertions, the algorithm enforces a total order function on them and computes a new position for those operations in order to achieve convergence. *YATA* also allows offline editing and uses garbage collection to retrieve a certain remove operation after a fixed period for all online users. However the authors mention that offline garbage collection is not supported by this work.

Our approach uses a peer-to-peer architecture and replicated data design to allow high-level collaboration between participants and the atomic change level is a line of code which is more suitable for a collaborative coding. We propose another IT function that ensures convergence.

5.4 IT function based on absolute positions

In the following, we consider that all the operations are not commutative even though it is possible that happens. We also state that the atomic change level is the line of source code. Then, we propose to add to the signature of the insert operation two new parameters, which are filled with the absolute position of the symbol [1] to be inserted and the identity of its site.

By absolute position, we mean the position of the line of code in case the deleted operations are ignored. For example, consider the sequence of operations $S = [Ins(2, "a = a + 1", s_1); Del(3); Del(1); Ins(4, "c = b + a", s_1)]$. The absolute position of the line of code $d = c + 1$ inserted by $Ins(6, d = c + 1, s_1)$ after executing S is $8 = 6 + 2$.

In *OT* approach with decentralized integration, when a site generates an operation, it is first executed locally then broadcast to other sites. Each site receiving this operation will, before its execution, transform it with regard to concurrent operations already executed on the site. The *IT* function proposed to achieve this transformation is depicted in Fig.5.4.

1. The symbol is a line of source code in the rest of the paper

Formally, the signature of an insert operation o consists of a position p, a symbol c to be inserted at position p, the absolute position ap of the inserted symbol and the identifier s of the site where the operation is generated (i.e., $o = Ins(p, c, ap, s)$).

When an insert operation $o = Ins(p, c, ap, s)$ is generated, its parameters ap and s are set to the absolute position of the symbol to be inserted and the identifier of the generated site, respectively. Then, the parameter ap of o is incremented whenever its transformation w.r.t. another operation results in incrementation of its position p. As for some other IT functions, site identifiers are compared to deal with conflicting insert operations (i.e., insert operations with identical absolute positions). Note that to deal with conflicting insert operations, in (Randolph et al., 2015), the authors used lexical values of symbols (characters). In case of conflicts, the symbol with smaller lexical value is placed before the others. However, using lexical values of symbols is not appropriate, as in our context, symbols stand for lines of codes. In our IT, in case of conflicts, the line of code of the site with smaller identifier will appear before the others.

$$\mathrm{IT}(Ins(p_1, c_1, ap_1, s_1), Ins(p_2, c_2, ap_2, s_2)) = \begin{cases} Ins(p_1, c_1, ap_1, s_1) & \text{if } \mathrm{ap}_1 < \mathrm{ap}_2 \ \vee (\mathrm{ap}_1 = \mathrm{ap}_2 \wedge \mathrm{s}_1 < \mathrm{s}_2) \\ Ins(p_1 + 1, c_1, ap_1 + 1, s_1) & \text{if } \mathrm{ap}_1 > \mathrm{ap}_2 \ \vee (\mathrm{ap}_1 = \mathrm{ap}_2 \wedge \mathrm{s}_1 > \mathrm{s}_2) \end{cases}$$

$$\mathrm{IT}(Ins(p_1, c_1, ap_1, s_1), Del(p_2)) = \begin{cases} Ins(p_1, c_1, ap_1, s_1) & \text{if } \mathrm{p}_1 \leq \mathrm{p}_2 \\ Ins(p_1 - 1, c_1, ap_1, s_1) & \text{otherwise} \end{cases}$$

$$\mathrm{IT}(Del(p_1), Ins(p_2, c_2, ap_2, s_2)) = \begin{cases} Del(p_1) & \text{if } \mathrm{p}_1 < \mathrm{p}_2 \\ Del(p_1 + 1) & \text{otherwise} \end{cases}$$

$$\mathrm{IT}(Del(p_1), Del(p_2)) = \begin{cases} Del(p_1) & \text{if } \mathrm{p}_1 < \mathrm{p}_2 \\ Del(p_1 - 1) & \text{if } \mathrm{p}_1 > \mathrm{p}_2 \\ Nop() & \text{otherwise} \end{cases}$$

Figure 5.4 The proposed IT algorithm

The following lemma establishes that the above IT adjusts appropriately the absolute position of the symbol of any insert operation $o_1 = (p_1, c_1, ap_1, s_1)$ whenever it is delayed to be executed after a sequence of concurrent operations S.

Lemma 1. *Let $o_1 = Ins(p_1, c_1, ap_1, s_1)$ be an insert operation, S be a sequence of operations concurrent to o_1 and $o'_1 = IT^*(o_1, S) = Ins(p'_1, c_1, ap'_1, s_1)$. Then ap'_1 is the absolute position where c_1 will be inserted in case the sequence S is executed before o_1.*

Démonstration. (by induction on the length of S)
Suppose that ap_1 is the absolute position of c_1 in case o_1 is executed before all operations of S. Let us show that the absolute position of c_1 is appropriately updated in case it is executed after the operation sequence S.
1) If $S = []$ then $o'_1 = o_1$ and $ap'_1 = ap_1$.
2) If $S = [o_2]$ then $o'_1 = IT(o_1, o_2)$ and, according to our IT, the different possible

relationships between p_1, p'_1 are : $p'_1 = p_1$, $p'_1 = p_1 + 1$ and $p'_1 = p_1 - 1$.
2.1) If $p'_1 = p_1$ then, according to our IT, $ap'_1 = ap_1$ and the deleted or inserted symbol by o_2 is located after the symbol inserted by o'_1. Therefore, the absolute position, where c_1 will be inserted after executing $[o_2]$, is ap_1.
2.2) If $p'_1 = p_1 + 1$ then, according to our IT, $ap'_1 = ap_1 + 1$ and o_2 is an insert operation whose symbol is inserted before position p'_1. Therefore, the absolute position, where c_1 will be inserted after executing $[o_2]$, is $ap_1 + 1$.
2.3) If $p'_1 = p_1 - 1$ then, according to our IT, $ap'_1 = ap_1$, o_2 is a delete operation and the symbol deleted is located before the symbol inserted by o'_1. Therefore, the absolute position, where c_1 will be inserted after executing $[o_2]$, is ap_1.
3) If $S = [o_2]; S'$ then $o'_1 = IT^*(IT(o_1, o_2), S')$. Let $o''_1 = IT(o_1, o_2) = Ins(p''_1, c_1, ap''_1, s_1)$. We have shown above that the absolute position, where c_1 will be inserted after executing $[o_2]$, is ap''_1.
To achieve the proof, it suffices to repeat the above process for $o'_1 = IT^*(o''_1, S')$ until reaching an empty sequence. □ □

Let us now show how to compute the parameter absolute position ap of an insert operation $o = Ins(p, c, ap, s)$. This parameter is computed at the generation of o. Note that, unlike the approach in (Oster et al., 2006), this parameter is computed without handling two distinct states (view and data models) at every site.

5.4.1 How to compute the absolute position of an insert operation ?

Let S be a sequence of operations executed at some site s before generating an insert operation $o = Ins(p, c, ap, s)$. We denote by $AP(S, p)$ the absolute position of p w.r.t. S, i.e., its corresponding position after executing S, in case the deleted position are kept but marked deleted. When o is generated, its parameter ap is set to $AP(S, p)$.

Definition 1. *$AP(S, p)$ is computed inductively as follows :*
$AP(S, p) =$

$$\begin{cases} p & \text{if } S = [] \\ AP(S', p) & \text{if } S = S' \bullet [Ins(p', c', ap', s')] \wedge p < p' \\ ap' & \text{if } S = S' \bullet [Ins(p', c', ap', s')] \wedge p = p' \\ AP(S', p-1) + 1 & \text{if } S = S' \bullet [Ins(p', c', ap', s')] \wedge p > p' \\ AP(S', p) & \text{if } S = S' \bullet [Del(p')] \wedge \ p < p' \\ AP(S', p+1) & \text{if } S = S' \bullet [Del(p')] \ \wedge \ p \geq p' \end{cases}$$

Intuitively, if S is empty, the absolute position of p w.r.t. S is p.
For $S = S' \bullet [Ins(p', c', ap', s')]$, after executing the sequence S, the symbol c' of

$Ins(p', c', ap', s')$ is at position p'. Then :
- If $p < p'$, the execution of $Ins(p', c', ap', s')$ does not affect the part before position p'. Therefore, the absolute position of p w.r.t. S is its absolute position w.r.t. S' (i.e., $AP(S, p) = AP(S', p)$.
- If $p = p'$, the symbol c' is inserted at position $p' = p$ and its absolute position is ap'.
- If $p > p'$, the symbol at position $p - 1$ before executing $Ins(p', c', ap', s')$ will be shifted to position p. Then, the absolute position of p w.r.t. S is the absolute position of $p - 1$ w.r.t. S' plus 1 (i.e., $AP(S, p) = AP(S', p - 1) + 1$), since the operation $Ins(p', c', ap', s')$ will shift to position p the symbol at position $p - 1$.
Finally, for $S = S' \bullet [Del(p')]$, the execution of $Del(p')$ will delete the symbol at position p'. Then :
- If $p < p'$, the execution of $Del(p')$ does not affect the part before position p' (i.e., $AP(S, p) = AP(S', p)$).
- Otherwise, the absolute position of p w.r.t. S is the absolute position of $p + 1$ w.r.t. S', as executing $Del(p')$ after S' will shift the symbol at position $p + 1$ to position p (i.e., $AP(S, p) = AP(S', p + 1)$).

5.4.2 Properties of the absolute positions

We have shown that our IT allows to adjust appropriately the absolute position of any insert operation o w.r.t. a sequence S of concurrent operations executed before o, in some site. We have also provided a computation procedure of absolute position of a position p w.r.t. a sequence of operations S (i.e. $AP(S, p)$). Lemma 2 establishes some relationships between two positions p_1 and p_2 and their absolute positions w.r.t. S.

Lemma 2. *Let S be an operation sequence executed on some document, p_1 and p_2 two positions in the resulting document. Then,*

$$(i)\ p_1 < p_2 \Rightarrow AP(S, p_1) < AP(S, p_2).$$

$$(ii)\ p_1 = p_2 \Leftrightarrow AP(S, p_1) = AP(S, p_2).$$

$$(iii)\ AP(S, p_1) < AP(S, p_2) \Rightarrow p_1 < p_2.$$

Démonstration. For $S = []$: Then, $AP(S, p_1) = p_1$ and $AP(S, p_2) = p_2$. Consequently, it holds that $p_1 < p_2 \Leftrightarrow AP(S, p_1) < AP(S, p_2)$, and $p_1 = p_2 \Leftrightarrow AP(S, p_1) = AP(S, p_2)$.
(i) : For $S = S' \bullet [o']$, suppose that $p_1 < p_2 \Rightarrow AP(S', p_1) < AP(S', p_2)$ and let us show that $p_1 < p_2 \Rightarrow AP(S, p_1) < AP(S, p_2)$. Based on the definition of AP, we consider 7 cases :
1) $p_1 < p_2 < p'$: $AP(S, p_1) = AP(S', p_1)$, $AP(S, p_2) = AP(S', p_2)$. Consequently, $p_1 < p_2 \Rightarrow AP(S, p_1) < AP(S, p_2)$.

2) $p_1 < p' = p_2$ and $o' = Ins(p', c', ap', s') : AP(S, p_1) = AP(S', p_1)$ and $AP(S, p_2) = AP(S', p_2)$. It follows that $AP(S, p_1) < AP(S, p_2)$.
3) $p_1 < p' < p_2$ and $o' = Ins(p', c', ap', s') : AP(S, p_1) = AP(S', p_1)$ and $AP(S, p_2) = AP(S', p_2 - 1) + 1$. Since $p_1 < p_2 - 1$, it follows that $AP(S, p_1) < AP(S, p_2)$.
4) $p_1 < p' \leq p_2$ and $o' = Del(p') : AP(S, p_1) = AP(S', p_1)$ and $AP(S, p_2) = AP(S', p_2+1)$. Since $p_1 < p_2 < p_2 + 1$, it follows that $AP(S, p_1) < AP(S, p_2)$.
5) $p' = p_1 < p_2$ and $o' = Ins(p', c', ap', s') : AP(S, p_1) = ap'$ and $AP(S, p_2) = AP(S', p_2 - 1) + 1$. Since $p' = p_1 \leq p_2 - 1$, it follows that $AP(S, p_1) < AP(S, p_2)$.
6) $p' < p_1 < p_2$ and $o' = Ins(p', c', ap', s') : AP(S, p_1) = AP(S', p_1-1)+1$ and $AP(S, p_2) = AP(S', p_2 - 1) + 1$. Since $p' - 1 = p_1 - 1 < p_2 - 1$, it follows that $AP(S, p_1) < AP(S, p_2)$.
7) $p' \leq p_1 < p_2$ and $o' = Del(p') : AP(S, p_1) = AP(S', p_1+1)$ and $AP(S, p_2) = AP(S', p_2+1)$. Since $p' < p_1 + 1 < p_2 + 1$, it follows that $AP(S, p_1) < AP(S, p_2)$.
(ii) : The proof of $(p_1 = p_2 \Rightarrow AP(S, p_1) = AP(S, p_2))$ is trivial. For $(AP(S, p_1) = AP(S, p_2) \Rightarrow p_1 = p_2)$, we suppose that $AP(S, p_1) = AP(S, p_2) \wedge p_1 \neq p_2$. According to (i), $p_1 < p_2 \Rightarrow AP(S, p_1) < AP(S, p_2)$ and $p_2 < p_1 \Rightarrow AP(S, p_2) < AP(S, p_1)$, which is in contradiction with the assumption $(AP(S, p_1) = AP(S, p_2) \wedge p_1 \neq p_2))$.
(iii) : Suppose that $AP(S, p_1) < AP(S, p_2)$ and $p_2 \leq p_1$. According to (i) and (ii), $p_2 \leq p_1$ implies that $AP(S, p_2) \leq AP(S, p_1)$, which is in contradiction with the assumption. □

□

These nice features are however not sufficient to ensure consistency, as, in the context of a decentralised OT approach, the operations of S may be executed (after transformation) in a different order in another site. Thus, it is necessary to prove that for equivalent sequences [2], the absolute position of any insert operation o is the same. This is the purpose of Theorem 1. Corollary 1, derived fromTheorem 1, states that the relationships established in Lemma 2 are still valid for equivalent sequences. It allows to prove the consistency of the OT approach proposed here.

Theorem 1. *Let S_1 and S_2 be two sequences which consist of the same set of original operations executed, after integration, in two different orders. Let p be a position. Then : $AP(S_1, p) = AP(S_2, p)$.*

Démonstration. By assumption, sequences S_1 and S_2 consist of the same set of original operations executed, after integration, in two different orders. The different execution orders can be obtained by successive pairwise permutations of concurrent operations. Without lost of generality, we can suppose that $S_1 = S \bullet [o_1; o_{21}] \bullet S'$ and $S_2 = S \bullet [o_2; o_{12}] \bullet S'$, where $o_{12} = IT(o_1, o_2)$ and $o_{21} = IT(o_2, o_1)$. To show that $AP(S_1, p) = AP(S_2, p)$, it suffices to prove that $AP(S \bullet [o_1; o_{21}], p) = AP(S \bullet [o_2; o_{12}], p)$. The basic transformation cases (k) w.r.t. our IT for $o_{12} = IT(o_1, o_2)$ and $o_{21} = IT(o_2, o_1)$ are reported in Table

2. sequences that consists of the same set of original operations executed, after integration, in different orders

5.1. The other cases are symmetrical to those in Table 5.1. Table 5.2 reports values of $AP(S \bullet [o_1; o_{21}], p)$ and $AP(S \bullet [o_2; o_{12}], p)$ for every transformation case. We have $AP(S_1, p) = AP(S_2, p)$ for all cases. □ □

Corollary 1. *Let S_1 and S_2 be two sequences which consist of the same set of original operations executed, after integration, in two different orders, p_1 and p_2 two positions in the resulting documents. Then,*

$$p_1 < p_2 \Leftrightarrow AP(S_1, p_1) < AP(S_2, p_2)$$

$$p_1 = p_2 \Leftrightarrow AP(S_1, p_1) = AP(S_2, p_2).$$

Démonstration. According to Lemma 2, $p_1 < p_2 \Leftrightarrow AP(S_1, p_1) < AP(S_1, p_2)$ and $p_1 = p_2 \Leftrightarrow AP(S_1, p_1) = AP(S_1, p_2)$. Using Theorem 1, we can state that $AP(S_1, p_1) = AP(S_2, p_1)$ and $AP(S_1, p_2) = AP(S_2, p_2)$. □ □

Table 5.1 Transformation cases of $IT(o_1, o_2)$ and $IT(o_2, o_1)$ for our IT

k	IT cases for $o_{12} = IT(o_1, o_2)$ and $o_{21} = IT(o_2, o_1)$
0	$o_1 = Ins(p_1, c_1, ap_1, s_1) \wedge o_2 = Ins(p_2, c_2, ap_2, s_2) \wedge ap_1 < ap_2$
1	$o_1 = Ins(p_1, c_1, ap_1, s_1) \wedge o_2 = Ins(p_2, c_2, ap_2, s_2) \wedge ap_1 = ap_2 \wedge s_1 < s_2$
2	$o_1 = Ins(p_1, c_1, ap_1, s_1) \wedge o_2 = Del(p_2) \wedge p_1 < p_2$
3	$o_1 = Ins(p_1, c_1, ap_1, s_1) \wedge o_2 = Del(p_2) \wedge p_1 = p_2$
4	$o_1 = Ins(p_1, c_1, ap_1, s_1) \wedge o_2 = Del(p_2) \wedge p_1 > p_2$
5	$o_1 = Del(p_1) \wedge o_2 = Del(p_2) \wedge p_1 < p_2$
6	$o_1 = Del(p_1) \wedge o_2 = Del(p_2) \wedge p_1 = p_2$

Table 5.2 Computing $AP(S \bullet [o_1; o_{21}], p)$ and $AP(S \bullet [o_2; o_{12}], p)$

Transformation cases	$AP(S \bullet [o_1; o_{21}], p)$	$AP(S \bullet [o_2; o_{12}], p)$
$k \in \{0, 1\}$ $ap_1 = ap_{12} \leq ap_2 = ap_{21} - 1$ $p_{12} = p_1 \leq p_2 = p_{21} - 1$	ap_1 if $p = p_1$ $AP(S, p)$ if $p < p_1$ $AP(S, p-1) + 1$ if $p_1 < p < p_{21}$ ap_{21} if $p = p_{21}$ $AP(S, p-2) + 2$ if $p \geq p_{21}$	ap_{12} if $p = p_1$ $AP(S, p)$ if $p \leq p_1$ $AP(S, p-1) + 1$ if $p_1 < p < p_{21}$ $ap_2 + 1$ if $p = p_{21}$ $AP(S, p-2) + 2$ if $p \geq p_{21}$
$k \in \{2, 3\}$ $p_{12} = p_1 \leq p_2 = p_{21} - 1$ $ap_{12} = ap_1$	$AP(S, p)$ if $p < p_1$ ap_1 if $p = p_1$ $AP(S, p-1) + 1$ if $p_1 < p < p_{21}$ $AP(S, p) + 1$ if $p \geq p_{21}$	$AP(S, p)$ if $p < p_1$ ap_{12} if $p = p_1$ $AP(S, p-1) + 1$ if $p_1 < p \leq p_{21}$ $AP(S, p) + 1$ if $p \geq p_{21}$
$k = 4$ $p_{12} + 1 = p_1 > p_2 = p_{21}$ $ap_{12} = ap_1 + 1$	$AP(S, p)$ if $p < p_2$ $AP(S, p+1)$ if $p_2 \leq p < p_{12}$ $ap_1 + 1$ if $p = p_{12}$ $AP(S, p) + 1$ if $p > p_{12}$	$AP(S, p)$ if $p < p_2$ $AP(S, p+1)$ if $p_2 \leq p < p_{12}$ ap_{12} if $p = p_{12}$ $AP(S, p) + 1$ if $p > p_{12}$
$k = 5$ $p_{12} = p_1 < p_2 = p_{21} + 1$	$AP(S, p)$ if $p < p_1$ $AP(S, p+1)$ if $p_1 \leq p < p_{21}$ $AP(S, p+2)$ if $p \geq p_{21}$	$AP(S, p)$ if $p < p_1$ $AP(S, p+1)$ if $p_1 \leq p < p_{21}$ $AP(S, p+2)$ if $p \geq p_{21}$
$k = 6$ $o_{12} = o_{21} = Nop() \wedge o_1 = o_2$	$AP(S \bullet [o_1], p)$	$AP(S \bullet [o_2], p)$

5.5 Proof of correctness

To prove consistency, it suffices to show that our IT satisfies both properties TP1 and TP2. As in (Boucheneb et al., 2010), we use difference bound matrices to handle symbolically

parameters of operations. This data structure is very useful to represent, manipulate symbolically the position and absolute position parameters of the update operations. It allows also to verify symbolically the properties TP1 and TP2.

5.5.1 Difference bound matrices

Let $X = \{x_1, ..., x_n\}$ be a finite and nonempty set of variables. An atomic constraint over X is a constraint of the form $x_i - x_j \prec c, x_i \prec c$, or $-x_j \prec c$, where $x_i, x_j \in X$, $\prec \in \{<, \leq\}$ and $c \in \mathbb{Z}$, $\mathbb{Z}$ being the set of integers. Constraints of the form $x_i - x_j \prec c$ are triangular constraints while the others are simple constraints. Constraints $x_i \prec x_j + c$, $x_i = x_j + c$, $x_i \geq x_j + c$, $x_i > x_j + c$, $x_i > c$, $-x_i > c$, $x_i \geq c$ and $-x_i \geq c$ are considered as abbreviations of atomic constraints. In our context, X is a set of nonnegative integer variables (discrete variables), representing operation parameters (positions and absolute positions of symbols). Therefore, atomic constraints $x_i - x_j < c, x_i < c$ and $-x_i < c$ are equivalent to $x_i - x_j \leq c - 1$, $x_i \leq c - 1$ and $-x_i \leq c - 1$, respectively. Moreover, we are interested in triangular constraints (i.e., all atomic constraints are supposed to be of the form $x_i - x_j \leq c$). In the rest of the paper, for simplicity, we will invariantly use atomic constraints or their abbreviations.

A difference bound matrix is used to represent a set of atomic constraints. Given a set of atomic constraints A over the set of variables X. The DBM of A is the square matrix M of order $|X|$, where m_{ij} is the upper bound of the difference $x_i - x_j$ in A. By convention $m_{ii} = 0$, for every $x_i \in X$. In case, there is no constraint in A on $x_i - x_j$ $(i \neq j)$, m_{ij} is set to ∞. For example, we report, in Table **??**, the DBM M of the following set of atomic constraints :

$$A = \{x_2 - x_1 \leq 5, x_1 - x_2 \leq -1, x_3 - x_1 \leq 3, x_1 - x_3 \leq 0\}.$$

Though the same nonempty domain may be expressed by different sets of atomic constraints, their DBMs have a unique form called *canonical form*. The canonical form of a DBM is the representation with tightest bounds on all differences between variables. It can be computed, in $O(n^3)$, n being the number of variables in the DBM, using a shortest-path algorithm, like Floyd-Warshall's all-pairs shortest-path algorithm (Behrmann et al., 2006). As an example, Table **??** shows the canonical form M' of the DBM M. Canonical forms make easier some operations over DBMs like the test of equivalence. Two sets of atomic constraints are equivalent iff the canonical forms of their DBMs are identical.

A set of atomic constraints may be inconsistent (i.e., its domain is empty). To verify the consistency of a set of atomic constraints, it suffices to apply a shortest-path algorithm and to stop the algorithm as soon as a negative cycle is detected (i.e., $m_{ii} < 0$, for some variable x_i). The presence of negative cycles means that the set of atomic constraints is

inconsistent.

Table 5.3 Some examples of DBMs

M	x_1	x_2	x_3
x_1	0	-1	0
x_2	5	0	∞
x_3	3	∞	0

M'	x_1	x_2	x_3
x_1	0	-1	0
x_2	5	0	5
x_3	3	2	0

Table 5.4 Symbolic transformations of $IT(o_1, o_2)$ and $IT(o_2, o_1)$ for our IT

k	A after both transformations
0	$A = \{p_1 < p_2, ap_1 < ap_2, p_{12} = p_1, p_{21} = p_2 + 1, ap_{12} = ap_1, ap_{21} = ap_2 + 1\}$ $o_{12} = Ins(p_{12}, c_1, ap_{12}, s_1)$, $o_{21} = Ins(p_{21}, c_2, ap_{21}, s_2)$
1	$A = \{p_1 = p_2, ap_1 = ap_2, s_1 < s_2, p_{12} = p_1, p_{21} = p_2 + 1, ap_{12} = ap_1, ap_{21} = ap_2 + 1\}$ $o_{12} = Ins(p_{12}, c_1, ap_{12}, s_1)$, $o_{21} = Ins(p_{21}, c_2, ap_{21}, s_2)$
2	$A = \{p_1 < p_2, p_{12} = p_1, p_{21} = p_2 + 1, ap_{12} = ap_1\}$ $o_{12} = Ins(p_{12}, c_1, ap_1, s_1)$, $o_{21} = Del(p_{21})$
3	$A = \{p_1 = p_2, p_{12} = p_1, p_{21} = p_2 + 1, ap_{12} = ap_1\}$ $o_{12} = Ins(p_{12}, c_1, ap_{12}, s_1)$, $o_{21} = Del(p_{21})$
4	$A = \{p_1 > p_2, p_{12} = p_1 - 1, p_{21} = p_2, ap_{12} = ap_1\}$ $o_{12} = Ins(p_{12}, c_1, ap_{12}, s_1)$, $o_{21} = Del(p_{21})$
5	$A = \{p_1 < p_2, p_{12} = p_1, p_{21} = p_2 - 1, ap_{12} = ap_1\}$ $o_{12} = Del(p_{12})$, $o_{21} = Del(p_{21})$
6	$A = \{p_1 = p_2\}$ $o_{12} = Nop()$, $o_{21} = Nop()$

Table 5.5 N.C.S. for $[o_1; o_{21}] \equiv [o_2; o_{12}]$

c_i	Kind of o_{12}	Kind of o_{21}	N.S.C. of convergence
c_0	Ins	Ins / Del	$p_{12} = p_1 \leq p_2 = p_{21} - 1$
c_1	Ins	Ins	$p_{21} = p_2 \leq p_1 = p_{12} - 1$
c_2	Del	Del	$p_{12} = p_1 < p_2 = p_{21} + 1$
c_3	Ins / Del	Del	$p_{21} = p_2 < p_1 = p_{12} + 1$
c_4	Ins	Ins	$p_{21} = p_{12} \wedge p_1 = p_2 \wedge c_1 = c_2$
c_5	Nop	Nop	$[o_1] \equiv [o_2]$

5.5.2 Verification of TP1

Let o_1 and o_2 be two concurrent operations generated on the same state. According to Corollary 1, in case $o_1 = Ins(p_1, c_1, ap_1, s_1)$ and $o_2 = Ins(p_1, c_1, ap_1, s_1)$, $p_1 < p_2 \Leftrightarrow ap_1 < ap_2$ and $p_1 = p_2 \Leftrightarrow ap_1 = ap_2$.

Let o_1 and o_2 be two concurrent operations, $o_{12} = IT(o_1, o_2)$ and $o_{21} = IT(o_2, o_1)$ their transformations w.r.t. o_2 and o_1, respectively. Our IT function satisfies property TP1 iff $[o_1; o_{21}] \equiv [o_2; o_{12}]$. We report in Tables 5.1 and 5.4, the transformation cases of our

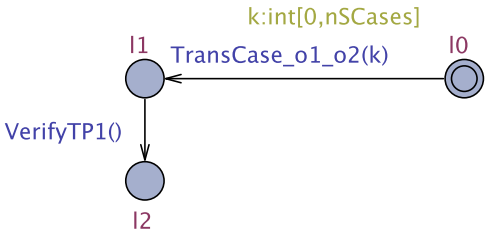


Figure 5.5 Automaton used to verify TP1

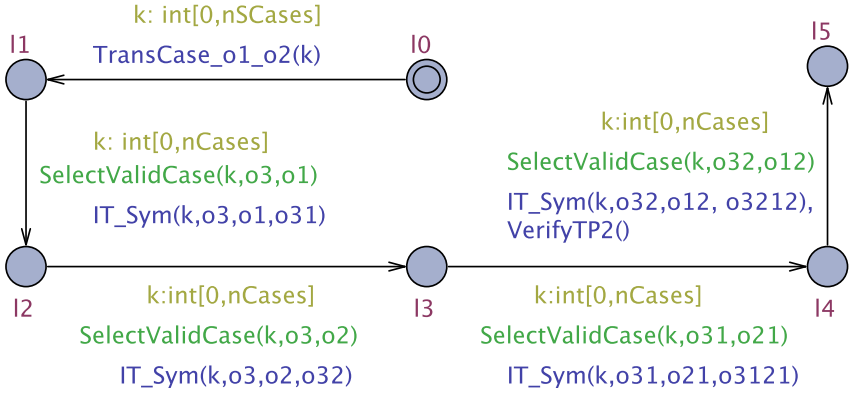


Figure 5.6 Automaton used to verify TP2

IT function for $o_{12} = IT(o_1, o_2)$ and $o_{21} = IT(o_2, o_1)$ and their corresponding symbolic transformations. Note that some transformation cases are equivalent w.r.t. property TP1. For example, transformation case 0 is equivalent to $o_1 = Ins(p_1, c_1, ap_1, s_1) \wedge o_2 = Ins(p_2, c_2, ap_2, s_2) \wedge ap_2 < ap_1$ w.r.t. property TP1. Moreover, cases where o_1 or o_2 is $Nop()$ satisfy trivially TP1. Indeed, if $o_1 = Nop()$ then $[o_1; o_{21}] \equiv [Nop(); IT(o_2, Nop())] \equiv [o_2]$ and $[o_2; o_{12}] \equiv [o_2; IT(Nop(), o_2)] \equiv [o_2]$. So, to verify TP1, it suffices to consider the 6 basic transformation cases in Table 5.1.

The model, as depicted in Fig. 5.5, is used to verify whether or not our IT function satisfies the property TP1. The model starts by selecting a transformation case k (see Table 5.1) and performs the corresponding transformations as shown in Table 5.4 (function $TransCase\_o1\_o2(k)$). Finally, it verifies TP1 for the considered transformation case by calling function $VerifyTP1()$. This function sets a boolean variable named $TP1$ to true iff all valuations of the DBM resulted from the transformation case satisfies one of the necessary and sufficient conditions reported in Table 5.5 (i.e., property TP1 is satisfied). The same procedure is repeated for each transformation case.

For example, the transformation case $k = 0$ consists in setting $o_i = Ins(p_i, c_i, ap_i, s_i)$, for $i \in \{1, 2, 12, 21\}$ and creating a DBM over $p_1, p_2, p_{12}, p_{21}, ap_1, ap_2, ap_{12}, ap_{21}$, whose set of constraints is $A = \{p_1 < p_2, ap_1 < ap_2, p_{12} = p_2, p_{21} = p_2 + 1, ap_{12} = ap_1, ap_{21} = ap_2\}$. Function $verifyTP1()$ verifies whether or not A satisfies at least a necessary and suffi-

cient conditions of Table 5.5, which is the case as A satisfies the necessary and sufficient condition c_0.

We have used the tool UPPAAL to verify whether or not the model satisfies the safety property $AG\ TP1$ [3], which means our IT satisfies TP1. UPPAAL concludes that the property is satisfied. Table 5.5 gives a necessary and sufficient condition for $[o_1; o_{21}] \equiv [o_2; o_{12}]$. Let us take an example in Fig. 5.7 to illustrate the satisfiability of the property TP1 for our IT function. The operation $Ins(p, l, ap)$ means that we insert the line of code l at position p (0 is the first position), ap is the absolute position of the line being inserted.

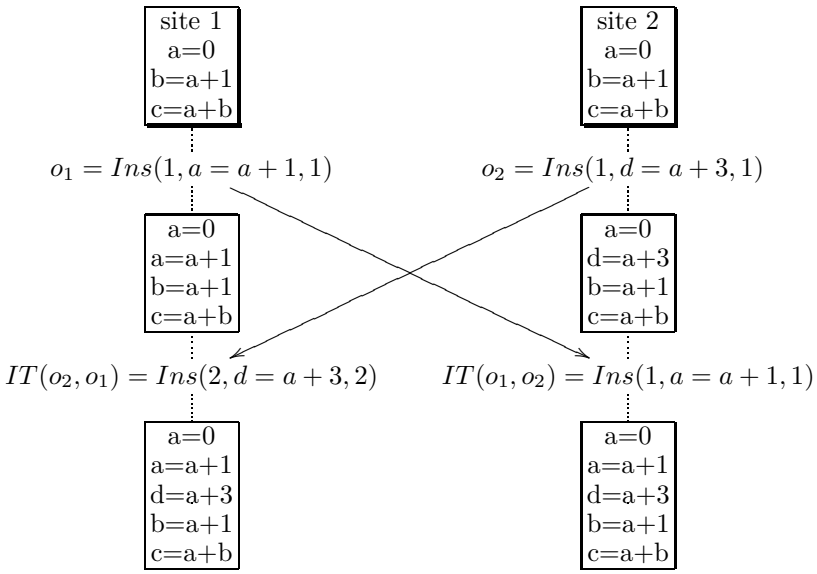


Figure 5.7 Satisfiability of TP1 for our IT.

5.5.3 Verification of TP2

Let o_1, o_2 and o_3 be three concurrent operations generated on the same state, $o_{12} = IT(o_1, o_2)$, $o_{21} = IT(o_2, o_1)$, $o_{31} = IT(o_3, o_1)$, and $o_{32} = IT(o_3, o_2)$. Our IT function satisfies property TP2 iff $IT(o_{31}, o_{21}) = IT(o_{32}, o_{12})$.

The model depicted at Fig. 5.6 is used to verify whether or not our IT function satisfies property TP2. The model starts by selecting a transformation case k (see Table 5.1) for $IT(o_1, o_2)$ and $IT(o_2, o_1)$ and then performing the corresponding transformations as for verifying property TP1. Afterwards, it computes, by considering successively all transformation cases, $o_{31} = IT(o_3, o_1)$, $o_{32} = IT(o_3, o_2)$, $o_{3121} = IT(o_{31}, o_{21})$ and $o_{3212} = IT(o_{32}, o_{3212})$. Finally, it verifies TP2 for the considered transformation cases by calling function $VerifyTP2()$. This function sets a boolean variable named $TP2$ to true iff $o_{3121} = o_{3212}$. Function $SelectValidCase(k, oi, oj, oij)$ verifies whether or not the selected transformation case k is valid, which means that there is in the current DBM at least a valuation s.t. the transformation case k can be applied to transform oi w.r.t. oj.

3. AG : along All paths Globally - $TP1$ has to hold inevitably along all paths

Function $IT\_Sym(k, oi, oj, oij)$ applies symbolically the operational transformation case k to oi w.r.t. oj.

We have used the tool UPPAAL to verify whether or not our model satisfies the safety property $AG\ TP2$. The property is not satisfied for some scenarios where there are at least two insert operations $o_i = Ins(p_i, c_i, ap_i)$ and $o_j = Ins(p_j, c_j, ap_j)$ generated at the same state s.t. Corollary 1 does not hold. We have shown in the previous subsection that such scenarios are impossible. When we exclude such scenarios, UPPAAL states that the property TP2 is satisfied (i.e., the proposed IT satisfies TP2).

For example, suppose that the transformations performed successively are $TransCase\_o1\_o2(0)$, $IT\_Sym(10, o3, o1, o31)$, $IT\_Sym(0, o3, o2, o32)$, $IT\_Sym(031, o21, o3121)$ and $IT\_Sym(9, o32, o21, o12)$. The evolution of sets of constraints of the DBM is shown in Table 5.6. Property TP2 is satisfied as $A = A \cup \{p_{3121} = p_{3212}, ap_{3121} = ap_{3212}\}$.

Table 5.6 Symbolic transformations of $IT(o_1, o_2)$ and $IT(o_2, o_1)$ for our IT

$Transformation$	Evolution of A
$TransCase\_o1\_o2(0)$	$A = \{p_1 < p_2, ap_1 < ap_2, p_{12} = p_1, p_{21} = p_2 + 1, ap_{12} = ap_1, ap_{21} = ap_2 + 1\}$ $o_1 = Ins(p_1, c_1, ap_1, s_1)$, $o_2 = Ins(p_2, c_2, ap_2, s_2)$ $o_{12} = Ins(p_{12}, c_1, ap_{12}, s_1)$, $o_{21} = Ins(p_{21}, c_2, ap_{21}, s_2)$
$IT\_Sym(10, o3, o1, o31)$	$A = A \cup \{p_3 = p_1, ap_3 = ap_1, s_3 > s_1, p_{31} = p_3 + 1, ap_{31} = ap_3 + 1\}$ $o_3 = Ins(p_3, c_3, ap_3, s_3)$
$IT\_Sym(0, o3, o2, o32)$	$A = A \cup \{p_3 < p_2, ap_3 < ap_2, p_{32} = p_3, ap_{32} = ap_3\}$ $o_{32} = Ins(p_{32}, c_3, ap_{32}, s_3)$
$IT\_Sym(0, o31, o21, o3121)$	$A = A \cup \{p_{31} < p_{21}, p_{3121} = p_{31}, ap_{3121} = ap_{31}\}$ $o_{3121} = Ins(p_{3121}, c_3, ap_{3121}, s_3)$
$IT\_Sym(9, o32, o21, o3212)$	$A = A \cup \{p_{32} = p_{21}, ap_{32} > ap_{21}, p_{3212} = p_{32} + 1, ap_{3212} = ap_{32} + 1\}$ $o_{3212} = Ins(p_{3212}, c_3, ap_{3212}, s_3)$

Let S_1 and S_2 be two sequences which consist of the same set of original operations executed, after integration, in two different orders. Let p_1 and p_2 be two positions. By definition, if $p_1 \leq p_2$ then $AP(S_1, p_1) \leq AP(S_1, p_2)$. Theorem 1 states that $AP(S_1, p_1) = AP(S_2, p_1)$, $AP(S_1, p_2) = AP(S_2, p_2)$, then :

1) $AP(S_1, p_1) \leq AP(S_2, p_2)$ iff $p_1 \leq p_2$
2) $AP(S_1, p_1) = AP(S_2, p_2)$ if $p_1 = p_2$.

The Fig. 5.8 below gives an example that satisfies the property TP2 for our IT function. By applying our IT algorithm in Fig. 5.4, we derive that $IT(IT(o_3, o_2), o_{12}) = IT(IT(Del(1), Ins(2, d = a + 3, 2)), o_{21}) = IT(Del(1), o_{12}) = IT(Del(1), Ins(1, a = a + 1, 2)) = Del(2)$. Similarly, $IT(IT(o_3, o_1), o_{21}) = IT(IT(Del(1), Ins(1, a = a+1, 1)), o_{21}) = IT(Del(2), o_{21}) = IT(Del(2), Ins(3, d = a + 3, 3)) = Del(2)$.

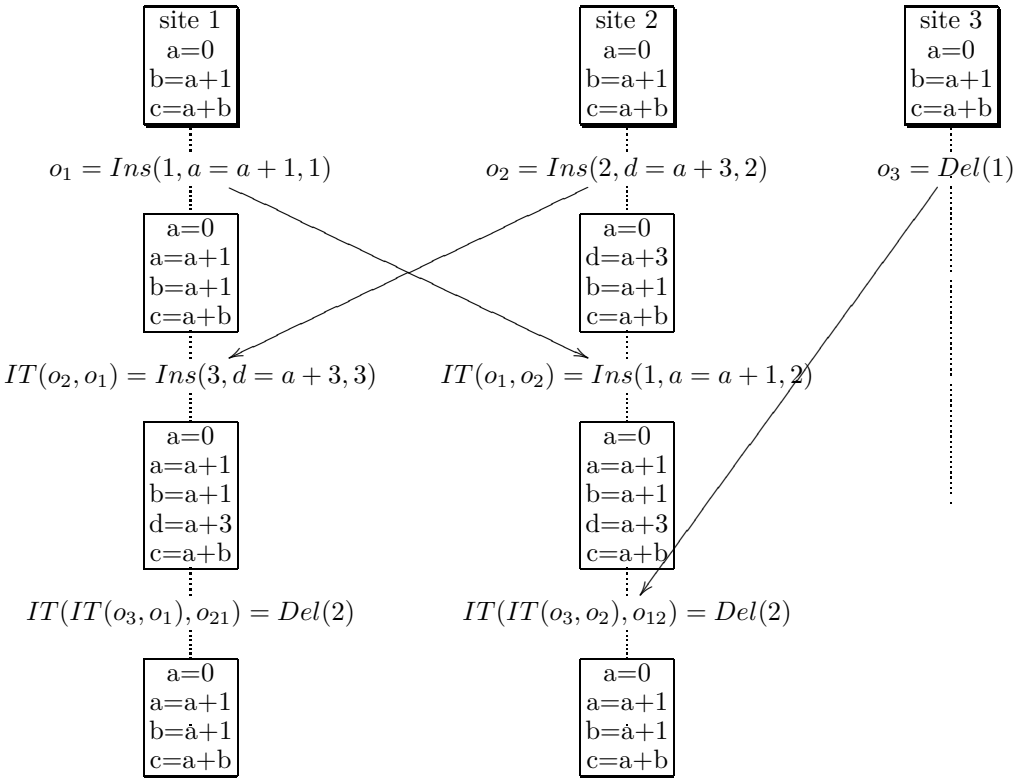


Figure 5.8 Satisfiability of TP2 for our IT.

5.6 Our paper limitations

As we can notice, our paper tackles the syntactic inconsistency problem due to concurrent updates on the same copy of a program source code. We do not address the semantic consistency of all the copies in this work. Indeed, if two programmers issue simultaneously the following lines of code : $a = a + 1$ from site 1 and $a = 2 * a$ from site 2, then the two semantically equivalent edits will be preserved in our IT function. However, this could be handled on a case-by-case basis since these kinds of inconsistencies are very hard problems. In the case of a conflict between two concurrent operations that involve the same variable (definition and use, called def-use situation), the priority is given to the "use" (read) operation. But if the two concurrent operations involve simultaneously def-use of the same variable, the operation issued by the site with the smaller identifier will appear before.

5.7 Conclusion and future work

In this paper, we have contributed a new operation transformation approach that is used in a collaborative coding work. We derived a IT function by adding to the signature of the insert operation two new parameters. The role of these two parameters is to respectively record the absolute position *ap* of the symbol to be inserted and the identity *s* of its site. These parameters are computed when the operation is generated and then the absolute

position parameter is updated whenever its transformation against another operation results in incrementation of its position p. We then proved that it maintains consistency properties *TP1* and *TP2* using model checking technique.

In future work, we are planning to apply our IT function to a specific peer-to-peer collaborative coding environment and derive the performance issue.

CHAPITRE 6 DISCUSSION GÉNÉRALE

Ce chapitre présente une syntèse de nos travaux et contributions ainsi qu'une analyse de nos résultats de recherche.

6.1 Synthèse des travaux

Les travaux présentés dans cette thèse ont commencé par une revue de littérature relative aux contrôles d'accès dans les systèmes collaboratifs des environnements distribués. Nous nous sommes focalisés sur la recherche de solutions pour répondre aux problématiques que pose cette catégorie de systèmes afin de d'assurer la consistance et la protection des données partagées à travers ces environnements. Le chapitre 2 nous a permis de passer en revue les différents modèles de contrôle d'accès, leurs avantages et inconvénients, mais également d'explorer la programmation orientée aspects pour mieux la comprendre afin de l'utiliser comme domaine d'application à la vérification de satisfiabilité des propriétés de sécurité en particulier. Un autre volet a été consacré à l'édition collaborative dans la programmation applicative dans laquelle nous avons exploré les différentes approches et algorithmes proposés dans la littérature pour faire face au problème de divergence des répliques. Cette revue de la littérature nous a permis de nous intéresser à certaines problématiques afin de circonscrire notre domaine de recherche dont les résultats ont fait l'objet de quatre articles parmi lesquels trois ont été insérés dans le présent document de thèse.

La première partie de notre recherche a été consacrée au renforcement des politiques de contrôle d'accès dans le partage de contenus multimédias accessibles au grand public. Pour cela, nous avons pris pour exemple l'encyclopédie Wikipédia qui est une plate-forme d'édition collaborative. Nous avons proposé l'utilisation du modèle de contrôle d'accès ABAC afin de renforcer et de fiabiliser les contenus qui sont publiés dans Wikipédia et réalisé une implémentation de notre solution. Cette partie a fait l'objet de deux articles dont l'un est inséré dans le présent document. Cet article propose une preuve formelle du modèle de contrôle d'accès proposé dans le premier.

La deuxième partie concerne l'utilisation d'une méthode formelle pour la vérification de propriétés notamment celles de sécurité dans un programme orienté aspects. Nous avons utilisé le solveur SMT $Z3$ et le graphe de flot de contrôle du programme pour nous assurer de la satisfiabilité d'un certain nombre de propriétés encapsulées dans les aspects pour faire face aux nombreuses interférences entre le programme de base et les préoccupations transverses pendant le tissage. La solution proposée se base sur l'exemple de la gestion des opérations d'un guichet automatique de banque et fait l'objet de l'article consigné dans le chapitre 4 de la présente thèse.

Quant au troisième volet, il est lui aussi consacré aux systèmes de codage collaboratif et aux preuves formelles dans lequel nous avons proposé un algorithme de transformation inclusive applicable à des systèmes d'édition de code source dans un environnement distribué. Cette fonction n'est autre que l'ajout dans la signature de l'opération d'insertion d'un nouveau paramètre : la "position absolue". Ce paramètre stocke la position de l'insertion d'une ligne de code en faisant abstraction des lignes supprimées précédemment. Une preuve formelle que cette fonction de transformation des opérations assure la convergence c'est-à-dire la satisfaction des propriétés $TP1$ et $TP2$ a été proposée en utilisant la technique de preuve symbolique. Ainsi, l'outil UPPAAL combiné avec les matrices de bornes a été utilisé pour les exécutions symboliques nécessaires à la confirmation de cette preuve. L'article qui découle de cette partie de notre recherche est présenté dans le chapitre 5 de notre thèse.

6.2 Analyse des résultats

Les résultats obtenus lors de cette thèse ont pour la plupart fait l'objet d'application dans un domaine précis. Pour ce qui concerne le renforcement de la politique de sécurité dans le partage de contenus multimédias, nous avons proposé une approche utilisant le modèle de contrôle d'accès ABAC afin d'appliquer une politique de contrôle d'accès garantissant la fiabilité des contenus publiés. Chaque contributeur gagne ou perd en réputation basée sur la fiabilité des contenus qu'il publie. En plus, un prototype de la solution a été développé pour montrer la faisabilité.

La fonction de transformation inclusive proposée pour répondre à la problématique de la convergence a, elle aussi, fait l'objet de preuve formelle. Notre fonction utilise un paramètre ajouté à la signature de l'opération d'insertion. La plupart des approches dans le codage collaboratif dans un environnement distribué basé sur un serveur central utilisent l'IT proposée par (Sun and Ellis, 1998). Cependant, cet algorithme n'assure pas la convergence dans un environnement fortement distribué qui ne dispose pas d'un serveur central.

CHAPITRE 7 CONCLUSION

Les systèmes d'édition collaborative permettent à des individus géographiquement dispersés dans le monde de travailler ensemble sur un même objet partagé (qui peut être un document, un projet, un code source, etc.). Ce partage pose souvent des problèmes de cohérence ou de fiabilité des contenus et le caractère dynamique de la collaboration et les interactions multiples des collaborateurs augmentent sensiblement les défis de cohérence et de fiabilité des données. Dans la présente thèse, nous les avons analysés et proposé des mécanismes à même de faciliter la détection d'erreurs, mais aussi de les prévenir. En guise de conclusion, nous présentons le sommaire des contributions de nos travaux ainsi que leurs limitations avant de proposer des esquisses de travaux futurs.

7.1 Sommaire des contributions

Nos travaux de recherche ont abouti à trois principales contributions.

La première est l'utilisation du modèle de contrôle d'accès ABAC pour renforcer la sécurité d'une plateforme de création et de partage de contenus (comparable à Wikipédia). Jusqu'à présent, le modèle ABAC a été que très rarement utilisé dans les environnements collaboratifs et jamais dans un environnement tel que celui que nous analysons. L'utilisation de ce modèle dans un environnement collaboratif nous a permis de voir qu'il est adapté aux situations où l'on a besoin de définir un contrôle d'accès au plus fin niveau afin de capter les attributs individuels des sujets, des objets, mais aussi du contexte évolutif de l'environnement. Une preuve formelle de la fiabilité du modèle de controle d'accès a été proposée à travers l'utilisation du solveur SMT $Z3$.

La deuxième contribution propose un mécanisme formel de vérification de propriétés notamment celles de sécurité dans un programme orienté aspects. En effet, ce paradigme attire la curiosité de la communauté des chercheurs, mais peine à être adopté par la communauté des développeurs de logiciels. Notre mécanisme combine l'utilisation du graphe de flot de contrôle du programme lors du tissage avec une formalisation à l'aide de solveurs SMT ($Z3$) pour valider la satisfaction des propriétés généralement encapsulées dans les aspects.

La dernière contribution consiste en la conception d'un algorithme de transformation inclusive qui garantit la convergence des répliques dans un contexte de codage collaboratif pour un environnement distribué. Il faut noter que jusqu'ici les recherches se sont focalisées sur la proposition d'approches permettant la collobaration dans le développement de logiciels, mais peu de recherches ont proposé des algorithmes de transformation opérationnelle et la preuve qu'ils assurent la convergence. Nous avons proposé une IT qui ajoute à la signature de l'opération d'insertion, un autre paramètre qui est la *position absolue* du

symbole au moment de son insertion, en faisant abstraction des symboles supprimés bien avant l'exécution de l'opération d'insertion. La preuve formelle de la convergence (c'est-à-dire la satisfaction des propriétés $TP1$ et $TP2$) a été démontrée dans cette contribution en utilisant l'outil *UPPAAL* et la preuve symbolique.

7.2 Limitations des travaux

Nos travaux de recherche comportent quelques limitations qui concernent particulièrement leur portée à grande échelle.

— Dans le chapitre 3, la limitation relevée concerne les contributeurs malveillants qui vont systématiquement taguer les autres contributeurs comme étant des vandales, ce qui pourrait biaiser un peu la réputation de certains contributeurs.

— Dans le chapitre 4, la deuxième contribution a pour limitation la portée de la solution. En effet, nous avons proposé notre solution en nous basant sur un exemple de code ayant, certes des fonctionnalités variées, mais il reste à produire un outil automatique pour le passage du graphe de flot de contrôle du programme à la spécification $Z3$. La limitation porte donc sur les applications complexes et de grande envergure, car elles peuvent générer un graphe avec beaucoup de noeuds et d'arcs.

— Dans le chapitre 5, la dernière contribution est accompagnée d'une limitation liée au fait que notre article concerne uniquement la problématique de l'incohérence syntaxique générée par les mises à jour concurrentes d'une même copie de code source. Nous n'avons pas couvert celle de l'incohérence sémantique, car ces problèmes sont très complexes. En effet, lorsque que deux programmeurs tentent simultanément d'insérer les lignes de code suivantes : $a = a + 1$ du site 1 et $a = 2 * a$ du site 2, même si ces deux lignes sont sémantiquement équivalentes, elles seront traitées comme deux lignes différentes, ce qui préserve quand même la cohérence des copies. Ces cas sont traités au cas par cas par les collaborateurs. Au vu des limitations ci-dessus énumérées, nous présentons, dans la prochaine section, les améliorations à venir.

7.3 Améliorations futures

Comme travaux futurs, nous prévoyons :

— dans la première contribution, d'utiliser d'autres techniques de vérification de modèles et de les comparer avec le solveur SMT $Z3$ en termes de performance ;

— dans la deuxième contribution, de développer un outil intégré qui pourra faire la transformation d'un graphe de flux de contrôle en $Z3$ afin de rendre la vérification utilisable même sur un programme orienté aspect plus complexe ;

— et, enfin, d'intégrer un modèle de contrôle d'accès dans les systèmes collaboratifs temps-réel.

RÉFÉRENCES

R. T. Alexander, J. M. Bieman, et A. A. Andrews, “Towards the systematic testing of aspect-oriented programs”, *Rapport technique, Colorado State University*, 2004.

C. Babu et H. R. Krishnan, “Fault model and test-case generation for the composition of aspects”, *ACM SIGSOFT Software Engineering Notes*, vol. 34, no. 1, pp. 1–6, 2009.

M. Badri, L. Badri, et M. Bourque-Fortin, “Generating unit test sequences for aspect-oriented programs : towards a formal approach using uml state diagrams”, dans *Information and Communications Technology, 2005. Enabling Technologies for the New Knowledge Society : ITI 3rd International Conference on.* IEEE, 2005, pp. 237–253.

J. Baltus, “La programmation orientée aspect et aspectj : Présentation et application dans un système distribué”, dans *Mini-Workshop : Systèmes Coopératifs. Matière Approfondie*, 2001.

H. Bani-Salameh, C. Jeffery, Z. Al-Sharif, et I. A. Doush, “Integrating collaborative program development and debugging within a virtual environment”, dans *International Workshop of Groupware.* Springer, 2008, pp. 107–120.

C. W. Barrett, R. Sebastiani, S. A. Seshia, C. Tinelli *et al.*, “Satisfiability modulo theories.” *Handbook of satisfiability*, vol. 185, pp. 825–885, 2009.

K. Beck et E. Gamma, *Extreme programming explained : embrace change.* addison-wesley professional, 2000.

G. Behrmann, P. Bouyer, K. G. Larsen, et R. Pelánek, “Lower and upper bounds in zone-based abstractions of timed automata”, *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 8, no. 3, pp. 204–215, 2006.

D. E. Bell, “Secure computer systems : A refinement of the mathematical model”, MITRE CORP MCLEAN VA, Rapp. tech., 1974.

D. E. Bell et L. J. LaPadula, “Secure computer systems : Mathematical foundations”, MITRE CORP BEDFORD MA, Rapp. tech., 1973.

L. Benmouffok, J. M. Marquès, M. Shapiro, P. Sutra, G. Tsoukalas *et al.*, “Telex : A semantic platform for cooperative application development”, dans *Conf. Française sur les Systèmes d’Exploitation (CFSE)*, 2009.

B. Berliner *et al.*, “Cvs ii : Parallelizing software development”, dans *Proceedings of the USENIX Winter 1990 Technical Conference*, vol. 341, 1990, p. 352.

M. L. Bernardi, “Reverse engineering of aspect oriented systems to support their comprehension, evolution, testing and assessment”, dans *Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on*. IEEE, 2008, pp. 290–293.

P. A. Bernstein et N. Goodman, “Multiversion concurrency control-theory and algorithms”, *ACM Transactions on Database Systems (TODS)*, vol. 8, no. 4, pp. 465–483, 1983.

K. J. Biba, “Integrity considerations for secure computer systems”, MITRE CORP BEDFORD MA, Rapp. tech., 1977.

H. Boucheneb et A. Imine, “On model-checking optimistic replication algorithms”, dans *Formal Techniques for Distributed Systems*. Springer, 2009, pp. 73–89.

H. Boucheneb, A. Imine, et M. Najem, “Symbolic model-checking of optimistic replication algorithms”, dans *International Conference on Integrated Formal Methods*. Springer, 2010, pp. 89–104.

J. K. Bowles, B. Bordbar, et M. Alwanain, “Weaving true-concurrent aspects using constraint solvers”, dans *Application of Concurrency to System Design (ACSD), 2016 16th International Conference on*. IEEE, 2016, pp. 35–44.

C. Bravo, R. Duque, J. Gallardo, J. García, et P. García, “A groupware system for distributed collaborative programming : usability issues and lessons learned”, dans *Proceedings for*, 2007, p. 6.

D. F. Brewer et M. J. Nash, “The chinese wall security policy”, dans *Security and privacy, 1989. proceedings., 1989 ieee symposium on*. IEEE, 1989, pp. 206–214.

J. Briffaut, X. Kauffmann-Tourkestansky, J.-F. Lalande, et W. W. Smari, “Generation of role based access control security policies for java collaborative applications”, dans *Emerging Security Information, Systems and Technologies, 2009. SECURWARE’09. Third International Conference on*. IEEE, 2009, pp. 224–229.

S. Bryant, P. Romero, et B. du Boulay, “Pair programming and the mysterious role of the navigator”, *International Journal of Human-Computer Studies*, vol. 66, no. 7, pp. 519–529, 2008.

M. Ceccato, P. Tonella, et F. Ricca, “Is aop code easier or harder to test than oop code”, dans *First Workshop on Testing Aspect-Oriented Program (WTAOP), Chicago, Illinois*, 2005.

A. Cockburn, *Crystal clear : a human-powered methodology for small teams*. Pearson Education, 2004.

B. Collins-Sussman, “The subversion project : buiding a better cvs”, *Linux Journal*, vol. 2002, no. 94, p. 3, 2002.

M. J. Covington, W. Long, S. Srinivasan, A. K. Dev, M. Ahamad, et G. D. Abowd, “Securing context-aware applications using environment roles”, dans *Proceedings of the sixth ACM symposium on Access control models and technologies*. ACM, 2001, pp. 10–20.

M. Dawande, M. Johar, S. Kumar, et V. S. Mookerjee, “A comparison of pair versus solo programming under different objectives : An analytical approach”, *Information Systems Research*, vol. 19, no. 1, pp. 71–92, 2008.

L. De Moura et N. Bjørner, “Z3 : An efficient smt solver”, dans *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

Y. Demchenko, L. Gommans, A. Tokmakoff, et R. V. Buuren, “Policy based access control in dynamic grid-based collaborative environment”, dans *Collaborative Technologies and Systems, 2006. CTS 2006. International Symposium on*. IEEE, 2006, pp. 64–73.

G. Denaro et M. Monga, “An experience on verification of aspect properties”, dans *Proceedings of the 4th international workshop on Principles of software evolution*. ACM, 2001, pp. 186–189.

P. Dewan, P. Agarwal, G. Shroff, et R. Hegde, “Distributed side-by-side programming”, dans *Proceedings of the 2009 ICSE workshop on cooperative and human aspects on software engineering*. IEEE Computer Society, 2009, pp. 48–55.

C. A. Ellis et S. J. Gibbs, “Concurrency control in groupware systems”, dans *Acm Sigmod Record*, vol. 18, no. 2. ACM, 1989, pp. 399–407.

T. Elrad, M. Aksit, G. Kiczales, K. Lieberherr, et H. Ossher, “Discussing aspects of aop”, *Communications of the ACM*, vol. 44, no. 10, pp. 33–38, 2001.

H. Fan et C. Sun, “Achieving integrated consistency maintenance and awareness in real-time collaborative programming environments : the coeclipse approach”, dans *Computer Supported Cooperative Work in Design (CSCWD), 2012 IEEE 16th International Conference on*. IEEE, 2012, pp. 94–101.

——, “Dependency-based automatic locking for semantic conflict prevention in real-time collaborative programming”, dans *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. ACM, 2012, pp. 737–742.

——, “Supporting semantic conflict prevention in real-time collaborative programming environments”, *ACM SIGAPP Applied Computing Review*, vol. 12, no. 2, pp. 39–52, 2012.

H. Fan, C. Sun, et H. Shen, “Atcope : any-time collaborative programming environment for seamless integration of real-time and non-real-time teamwork in software development”, dans *Proceedings of the 17th ACM international conference on Supporting group work*. ACM, 2012, pp. 107–116.

D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, et R. Chandramouli, “Proposed nist standard for role-based access control”, *ACM Transactions on Information and System Security (TISSEC)*, vol. 4, no. 3, pp. 224–274, 2001.

P. Fradet et S. Hong Tuan Ha, “Aspects of availability : Enforcing timed properties to prevent denial of service”, *Science of Computer Programming*, vol. 75, no. 7, pp. 516–542, 2010.

I. G. Franchin, O. A. L. Lemos, et P. C. Masiero, “Pairwise structural testing of object and aspect-oriented java programs”, dans *The 21th Software Engineering Brazilian Symposium, Joao Pessoa, PB, Brazil*, 2007.

N. Fraser, “Differential synchronization”, dans *Proceedings of the 9th ACM symposium on Document engineering*. ACM, 2009, pp. 13–20.

Y. Fu, J. Ding, et P. Bording, “An approach for modeling and analyzing crosscutting concerns”, dans *Service Operations, Logistics and Informatics, 2009. SOLI’09. IEEE/INFORMS International Conference on*. IEEE, 2009, pp. 91–97.

C. K. Georgiadis, I. Mavridis, G. Pangalos, et R. K. Thomas, “Flexible team-based access control using contexts”, dans *Proceedings of the sixth ACM symposium on Access control models and technologies*. ACM, 2001, pp. 21–27.

M. Goldman, G. Little, et R. C. Miller, “Real-time collaborative coding in a web ide”, dans *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 2011, pp. 155–164.

K. W. Hamlen et M. Jones, “Aspect-oriented in-lined reference monitors”, dans *Proceedings of the third ACM SIGPLAN workshop on Programming languages and analysis for security*. ACM, 2008, pp. 11–20.

M. A. Harrison, W. L. Ruzzo, et J. D. Ullman, “Protection in operating systems”, *Communications of the ACM*, vol. 19, no. 8, pp. 461–471, 1976.

L. Hattori et M. Lanza, “Syde : a tool for collaborative software development”, dans

Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2. ACM, 2010, pp. 235–238.

R. Hegde et P. Dewan, "Connecting programming environments to support ad-hoc collaboration", dans *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 2008, pp. 178–187.

M. P. Herlihy et J. M. Wing, "Linearizability : A correctness condition for concurrent objects", *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 12, no. 3, pp. 463–492, 1990.

M. N. Hossain, W. Kahl, et T. Maibaum, "A graph transformation approach to introducing aspects into software architectures." dans *MODELS (Satellite Events)*, 2017, pp. 54–63.

J. Hu, K. M. Khan, Y. Zhang, Y. Bai, et R. Li, "Role updating in information systems using model checking", *Knowledge and Information Systems*, vol. 51, no. 1, pp. 187–234, 2017.

V. C. Hu, D. Ferraiolo, et D. R. Kuhn, *Assessment of access control systems*. US Department of Commerce, National Institute of Standards and Technology, 2006.

A. Imine, P. Molli, G. Oster, et M. Rusinowitch, "Proving correctness of transformation functions in real-time groupware", dans *ECSCW'03*, Helsinki, Finland, 14.-18. September 2003, pp. 277 – 293.

D. Jackson, *Software Abstractions : logic, language, and analysis*. MIT press, 2012.

T. Jaeger et A. Prakash, "Requirements of role-based access control for collaborative systems", dans *Proceedings of the first ACM Workshop on Role-based access control*. ACM, 1996, p. 16.

G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, et J. Irwin, "Aspect-oriented programming", dans *European conference on object-oriented programming*. Springer, 1997, pp. 220–242.

G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, et W. G. Griswold, "An overview of aspectj", dans *ECOOP 2001—Object-Oriented Programming*. Springer, 2001, pp. 327–354.

J. Kolter, R. Schillinger, et G. Pernul, "A privacy-enhanced attribute-based access control system", dans *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 2007, pp. 129–143.

A. Kurniawan, C. Soesanto, et J. E. C. Wijaya, “Coder : Real-time code editor application for collaborative programming”, *Procedia Computer Science*, vol. 59, pp. 510–519, 2015.

L. Lamport, “Time, clocks, and the ordering of events in a distributed system”, *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.

B. W. Lampson, “Protection”, *ACM SIGOPS Operating Systems Review*, vol. 8, no. 1, pp. 18–24, 1974.

M. Lanza, L. Hattori, et A. Guzzi, “Supporting collaboration awareness with real-time visualization of development activity”, dans *2010 14th European Conference on Software Maintenance and Reengineering*. IEEE, 2010, pp. 202–211.

J. Lautamäki, A. Nieminen, J. Koskinen, T. Aho, T. Mikkonen, et M. Englund, “Cored : browser-based collaborative real-time editor for java web applications”, dans *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*. ACM, 2012, pp. 1307–1316.

O. A. L. Lemos et P. C. Masiero, “A pointcut-based coverage analysis approach for aspect-oriented programs”, *Information Sciences*, vol. 181, no. 13, pp. 2721–2746, 2011.

O. A. L. Lemos, A. M. R. Vincenzi, J. C. Maldonado, et P. C. Masiero, “Control and data flow structural testing criteria for aspect-oriented programs”, *Journal of Systems and Software*, vol. 80, no. 6, pp. 862–882, 2007.

D. Li et R. Li, “Preserving operation effects relation in group editors”, dans *Proceedings of the 2004 ACM conference on Computer supported cooperative work*. ACM, 2004, pp. 457–466.

M. Mankai et L. Logrippo, “Access control policies : Modeling and validation”, dans *5th NOTERE Conference (Nouvelles Technologies de la Répartition)*, 2005, pp. 85–91.

P. Massicotte, M. Badri, et L. Badri, “Generating aspects-classes integration testing sequences a collaboration diagram based strategy”, dans *Software Engineering Research, Management and Applications, 2005. Third ACIS International Conference on*. IEEE, 2005, pp. 30–37.

J. Michaux, X. Blanc, M. Shapiro, et P. Sutra, “A semantically rich approach for collaborative model edition”, dans *Proceedings of the 2011 ACM Symposium on Applied Computing*. ACM, 2011, pp. 1470–1475.

S. Mondal, S. Sural, et V. Atluri, “Towards formal security analysis of gtrbac using timed automata”, dans *Proceedings of the 14th ACM symposium on Access control models and technologies*. ACM, 2009, pp. 33–42.

A. Mougenot, X. Blanc, et M.-P. Gervais, “D-praxis : A peer-to-peer collaborative model editing framework”, dans *IFIP International Conference on Distributed Applications and Interoperable Systems*. Springer, 2009, pp. 16–29.

E. W. Myers, “Ano (nd) difference algorithm and its variations”, *Algorithmica*, vol. 1, no. 1-4, pp. 251–266, 1986.

P. Nasirifard, V. Peristeras, et S. Decker, “An annotation-based access control model and tools for collaborative information spaces”, dans *Emerging Technologies and Information Systems for the Knowledge Society*. Springer, 2008, pp. 51–60.

P. Nicolaescu, K. Jahns, M. Derntl, et R. Klamma, “Near real-time peer-to-peer shared editing on extensible data types”, dans *Proceedings of the 19th International Conference on Supporting Group Work*. ACM, 2016, pp. 39–49.

G. Oster, P. Molli, P. Urso, et A. Imine, “Tombstone transformation functions for ensuring consistency in collaborative editing systems”, *International Conference on Collaborative Computing : Networking, Applications and Worksharing (CollaborateCom)*, pp. 1–10, Nov 2006.

R. M. Parizi et A. A. A. Ghani, “Ajcfgraph-aspectj control flow graph builder for aspect-oriented software”, *International Journal of Computer Science*, vol. 3, pp. 170–181, 2008.

K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, et A. J. Demers, “Flexible update propagation for weakly consistent replication”, dans *ACM SIGOPS Operating Systems Review*, vol. 31, no. 5. ACM, 1997, pp. 288–301.

C. M. Pilato, B. Collins-Sussman, et B. W. Fitzpatrick, *Version Control with Subversion : Next Generation Open Source Version Control*. " O’Reilly Media, Inc.", 2008.

L. Prechelt, U. Stärk, et S. Salinger, “7 types of cooperation episodes in side-by-side programming”, 2008.

N. Preguica, J. M. Marques, M. Shapiro, et M. Letia, “A commutative replicated data type for cooperative editing”, dans *Distributed Computing Systems, 2009. ICDCS’09. 29th IEEE International Conference on*. IEEE, 2009, pp. 395–403.

A. Randolph, H. Boucheneb, A. Imine, et A. Quintero, “On synthesizing a consistent operational transformation approach”, *IEEE Transactions on Computers*, vol. 64, no. 4, pp. 1074–1089, 2015.

M. Reith, J. Niu, et W. H. Winsborough, “Toward practical analysis for trust management policy”, dans *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*. ACM, 2009, pp. 310–321.

X. Ren, F. Shah, F. Tip, B. G. Ryder, et O. Chesley, “Chianti : a tool for change impact analysis of java programs”, dans *ACM Sigplan Notices*, vol. 39, no. 10. ACM, 2004, pp. 432–448.

M. Ressel, D. Nitsche-Ruhland, et R. Gunzenhäuser, “An integrating, transformation-oriented approach to concurrency control and undo in group editors”, dans *Proceedings of the 1996 ACM conference on Computer supported cooperative work*. ACM, 1996, pp. 288–297.

B. G. Ryder et F. Tip, “Change impact analysis for object-oriented programs”, dans *Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*. ACM, 2001, pp. 46–53.

S. Salinger, C. Oezbek, K. Beecher, et J. Schenk, “Saros : an eclipse plug-in for distributed party programming”, dans *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering*. ACM, 2010, pp. 48–55.

R. S. Sandhu, “Lattice-based access control models”, *Computer*, no. 11, pp. 9–19, 1993.

R. S. Sandhu, E. J. Coyne, H. L. Feinstein, et C. E. Youman, “Role-based access control models”, *Computer*, no. 2, pp. 38–47, 1996.

A. Sarma, Z. Noroozi, et A. Van Der Hoek, “Palantír : raising awareness among configuration management workspaces”, dans *Software Engineering, 2003. Proceedings. 25th International Conference on*. IEEE, 2003, pp. 444–454.

A. Schaad et J. D. Moffett, “A lightweight approach to specification and analysis of role-based access control extensions”, dans *Proceedings of the seventh ACM symposium on Access control models and technologies*. ACM, 2002, pp. 13–22.

A. Schaad, V. Lotz, et K. Sohr, “A model-checking approach to analysing organisational controls in a loan origination process”, dans *Proceedings of the eleventh ACM symposium on Access control models and technologies*. ACM, 2006, pp. 139–149.

D. Shao, S. Khurshid, et D. E. Perry, “Evaluation of semantic interference detection in parallel changes : an exploratory experiment”, dans *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*. IEEE, 2007, pp. 74–83.

H. Shen et C. Sun, “Recipe : a prototype for internet-based real-time collaborative programming”, dans *Proceedings of the 2nd Annual International Workshop on Collaborative Editing Systems. Philadelphia, Pennsylvania, USA*. Citeseer, 2000.

W. W. Smari, J. Zhu, et P. Clemente, “Trust and privacy in attribute based access control for collaboration environments”, dans *Proceedings of the 11th International Conference*

on Information Integration and Web-based Applications & Services. ACM, 2009, pp. 49–55.

V. Stolz et E. Bodden, "Temporal assertions using aspectj", *Electronic Notes in Theoretical Computer Science*, vol. 144, no. 4, pp. 109–124, 2006.

M. Suleiman, M. Cart, et J. Ferrié, "Serialization of concurrent operations in a distributed collaborative environment", dans *ACM GROUP'97*, November 1997, pp. 435–445.

C. Sun et C. Ellis, "Operational transformation in real-time group editors : issues, algorithms, and achievements", dans *Proceedings of the 1998 ACM conference on Computer supported cooperative work*. ACM, 1998, pp. 59–68.

C. Sun, X. Jia, Y. Zhang, Y. Yang, et D. Chen, "Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems", *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 5, no. 1, pp. 63–108, 1998.

Y. Tahara, A. Ohsuga, et S. Honiden, "Formal verification of dynamic evolution processes of uml models using aspects", dans *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2017 IEEE/ACM 12th International Symposium on*. IEEE, 2017, pp. 152–162.

R. K. Thomas, "Team-based access control (tmac) : a primitive for applying role-based access controls in collaborative environments", dans *Proceedings of the second ACM workshop on Role-based access control*. ACM, 1997, pp. 13–19.

R. K. Thomas et R. S. Sandhu, "Task-based authorization controls (tbac) : A family of models for active and enterprise-oriented authorization management", dans *Database Security XI*. Springer, 1998, pp. 166–181.

M. Toahchoodee et I. Ray, "Validation of policy integration using alloy", *Lecture notes in computer science*, vol. 3816, p. 420, 2005.

W. Tolone, G.-J. Ahn, T. Pai, et S.-P. Hong, "Access control in collaborative systems", *ACM Computing Surveys (CSUR)*, vol. 37, no. 1, pp. 29–41, 2005.

N. Ubayashi et T. Tamai, "Aspect-oriented programming with model checking", dans *Proceedings of the 1st international conference on Aspect-oriented software development*. ACM, 2002, pp. 148–154.

A. van Deursen, A. Mesbah, B. Cornelissen, A. Zaidman, M. Pinzger, et A. Guzzi, "Adinda : a knowledgeable, browser-based ide", dans *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*. ACM, 2010, pp. 203–206.

N. Vidot, M. Cart, J. Ferrié, et M. Suleiman, “Copies convergence in a distributed real-time collaborative environment”, dans *Proceedings of the 2000 ACM conference on Computer supported cooperative work*. ACM, 2000, pp. 171–180.

Q. Wang, H. Jin, et N. Li, “Usable access control in collaborative environments : Authorization based on people-tagging”, dans *Computer Security–ESORICS 2009*. Springer, 2009, pp. 268–284.

F. Wedyan et S. Ghosh, “A dataflow testing approach for aspect-oriented programs”, dans *High-Assurance Systems Engineering (HASE), 2010 IEEE 12th International Symposium on*. IEEE, 2010, pp. 64–73.

J. Wloka, B. Ryder, F. Tip, et X. Ren, “Safe-commit analysis to facilitate team software development”, dans *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*. IEEE, 2009, pp. 507–517.

D. Xu, I. Alsmadi, et W. Xu, “Model checking aspect-oriented design specification”, dans *Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International*, vol. 1. IEEE, 2007, pp. 491–500.

D. Xu, O. El-Ariss, W. Xu, et L. Wang, “Testing aspect-oriented programs with finite state machines”, *Software Testing, Verification and Reliability*, vol. 22, no. 4, pp. 267–293, 2012.

J. Zhao, “Data-flow-based unit testing of aspect-oriented programs”, dans *Computer Software and Applications Conference, 2003. COMPSAC 2003. Proceedings. 27th Annual International*. IEEE, 2003, pp. 188–197.

W. Zhou et C. Meinel, “Team and task based rbac access control model”, dans *Network Operations and Management Symposium, 2007. LANOMS 2007. Latin American*. IEEE, 2007, pp. 84–94.